US 20010049697A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2001/0049697 A1**

JOHNDREW et al. (43) **Pub. Date: Dec. 6, 2001**

(54) **SYSTEM AND METHOD FOR RETRIEVING SOFTWARE RELEASE INFORMATION**

(76) Inventors: **THOMAS M. JOHNDREW**, APTOS, CA (US); **SANJEEV K. GUPTA**, SANTA CLARA, CA (US)

Correspondence Address:
D'ALESSANDRO & RITCHIE
P O BOX 640640
SAN JOSE, CA 95164-0640 (US)

(*) Notice: This is a publication of a continued prosecution application (CPA) filed under 37 CFR 1.53(d).

(21) Appl. No.: **09/058,943**

(22) Filed: **Apr. 10, 1998**

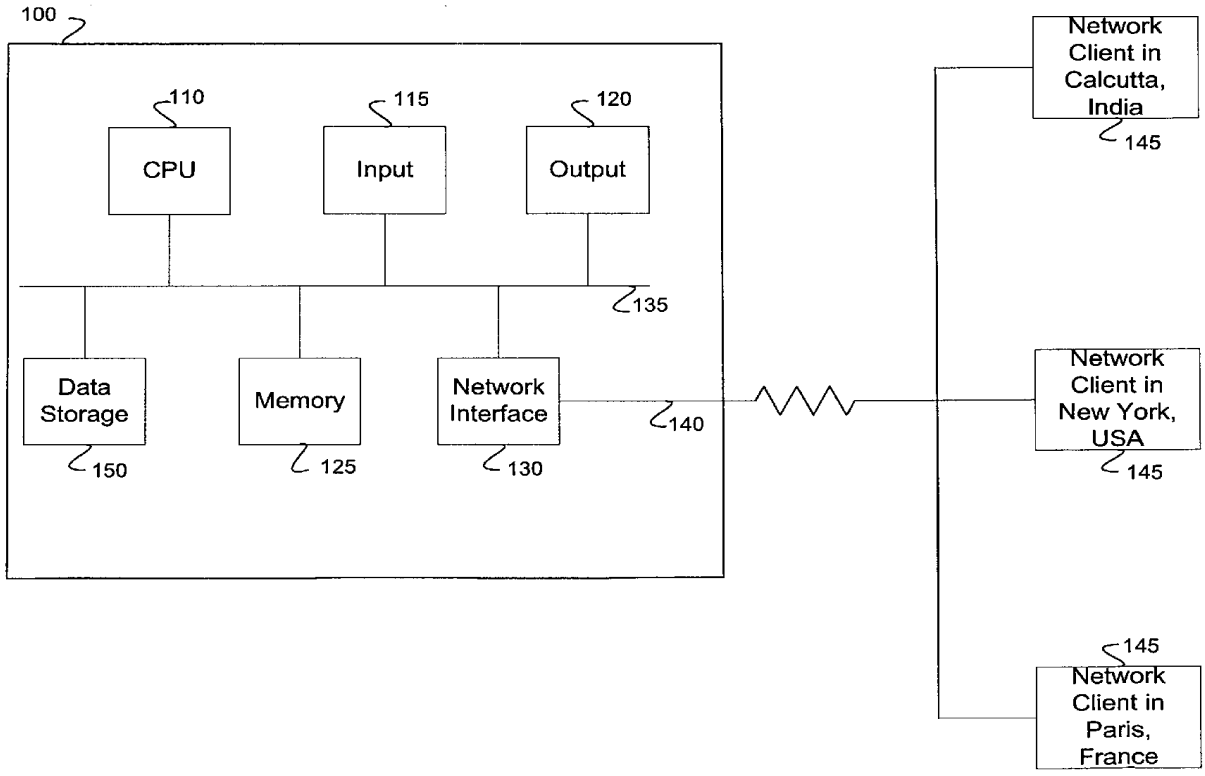
Publication Classification

(51) **Int. Cl.**[7] **G06F 15/00**

(52) **U.S. Cl.** **707/500**

(57) **ABSTRACT**

A method for retrieving software release information. A first step obtains a software defect data set, a second step obtains a software release data set, and a third step obtains a software release schedule data set. A fourth step relates at least two of the data sets to create an organized data set. A final step displays the contents of the organized data set thereby enabling a user to retrieve software release information.



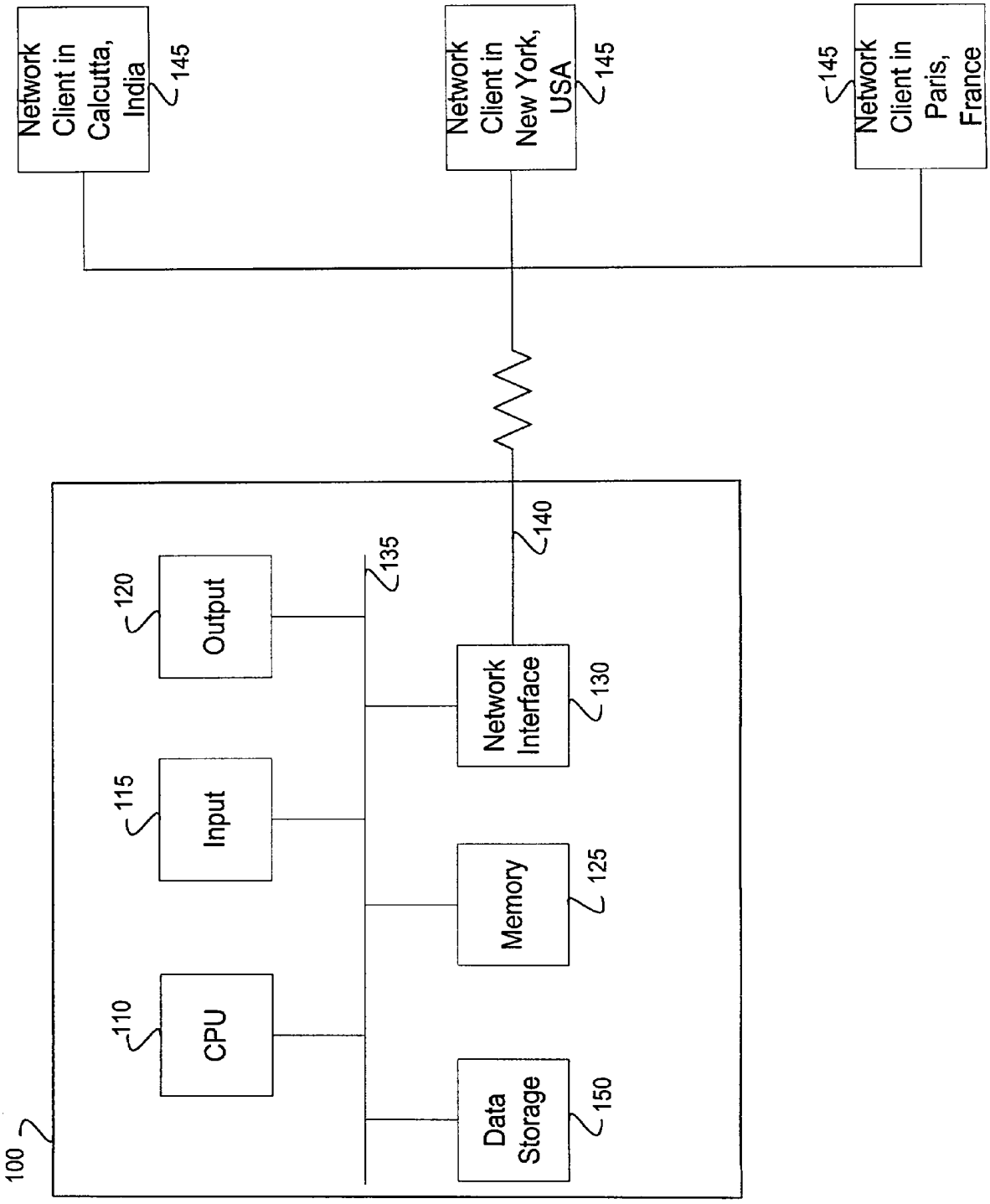

Network Client in Calcutta, India
145
Network Client in New York, USA
145
145
Network Client in Paris, France
140
135
120
Output
Network Interface
130
115
Input
Memory
125
110
CPU
Data Storage
150
100

FIG. 1

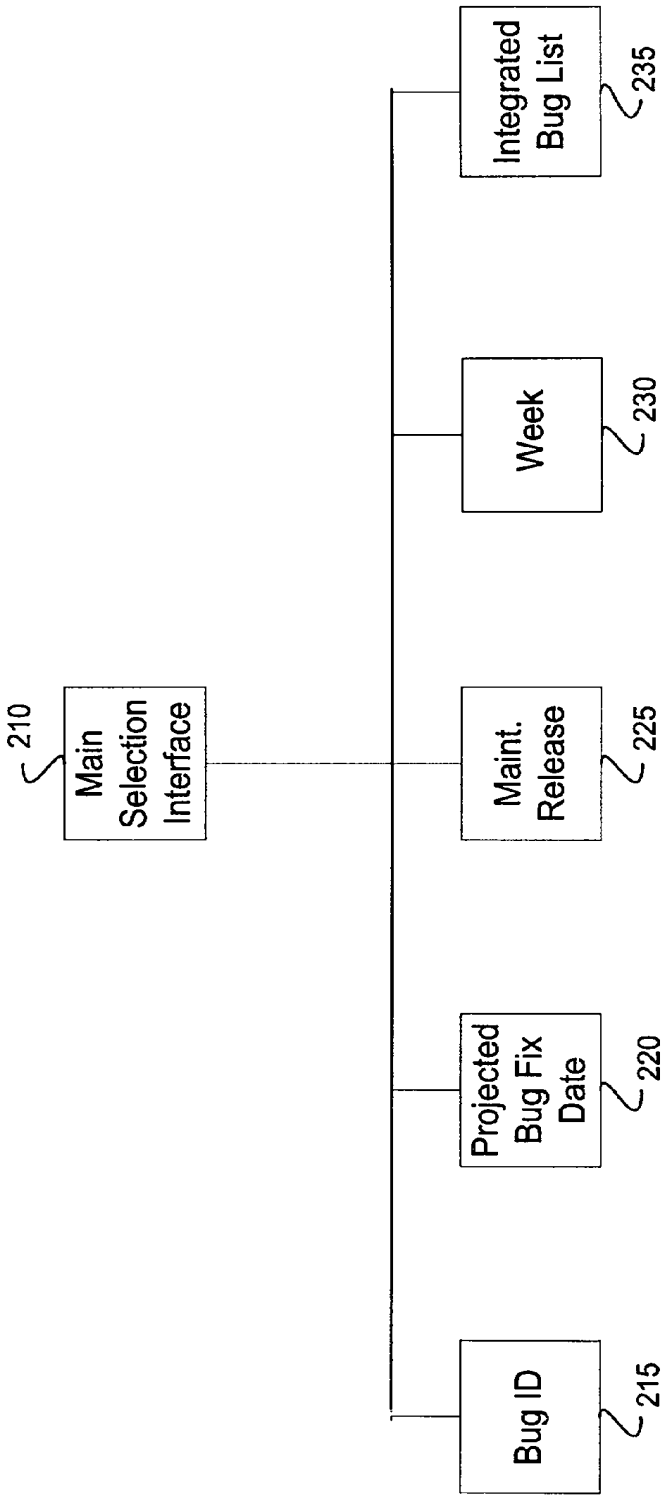

Main Selection Interface
210
Bug ID
215
Projected Bug Fix Date
220
Maint. Release
225
Week
230
Integrated Bug List
235
200

FIG. 2

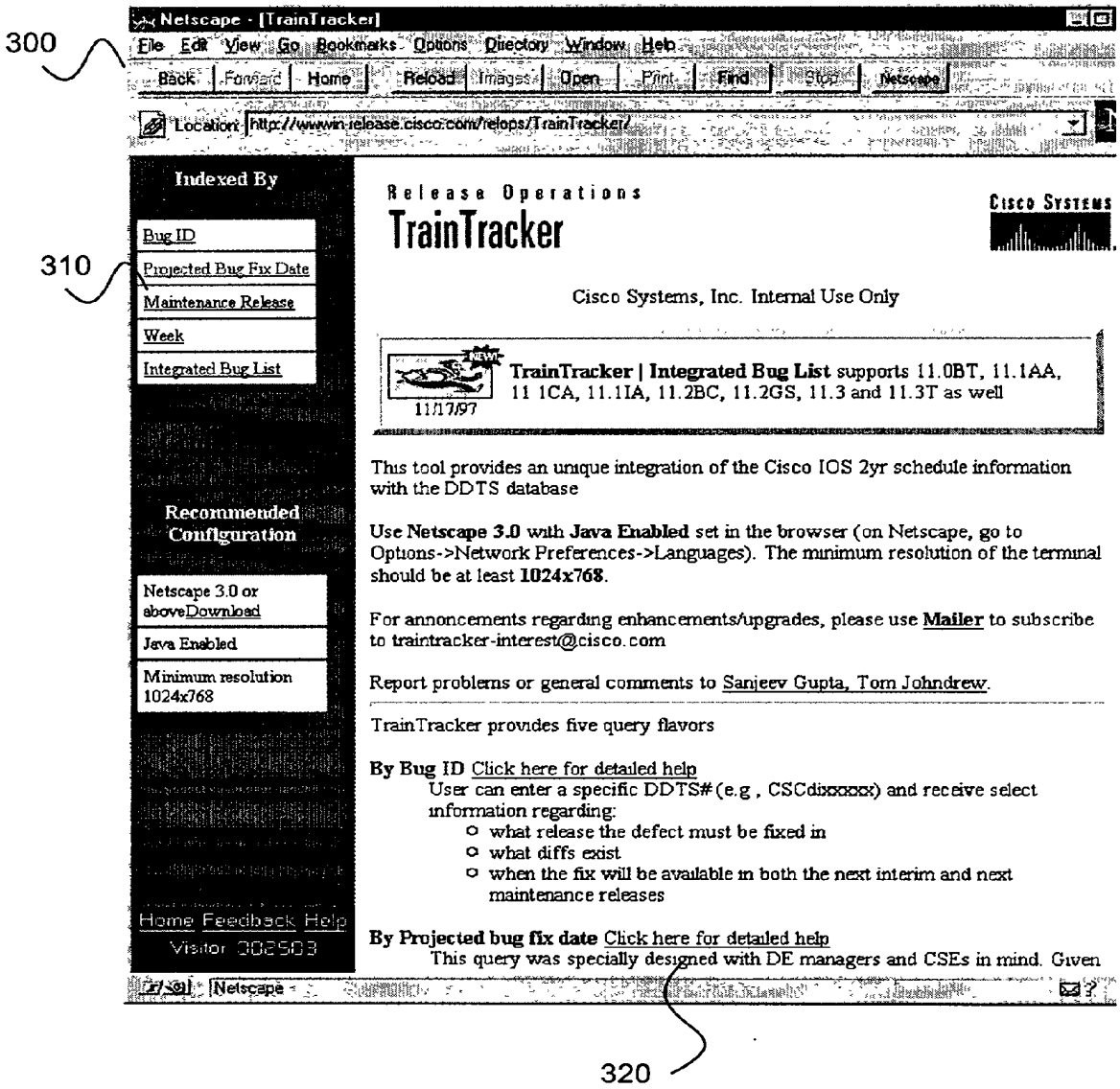

300
Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www-release.cisco.com/relops/TrainTracker/
Indexed By
Bug ID
310
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
Recommended Configuration
Netscape 3.0 or aboveDownload
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 002503
Release Operations
TrainTracker
Cisco Systems
Cisco Systems, Inc. Internal Use Only
NEW!
11/17/97
TrainTracker | Integrated Bug List supports 11.0BT, 11.1AA, 11 1CA, 11.1IA, 11.2BC, 11.2GS, 11.3 and 11.3T as well
This tool provides an unique integration of the Cisco IOS 2yr schedule information with the DDTS database
Use Netscape 3.0 with Java Enabled set in the browser (on Netscape, go to Options->Network Preferences->Languages). The minimum resolution of the terminal should be at least 1024x768.
For annoncements regarding enhancements/upgrades, please use Mailer to subscribe to traintracker-interest@cisco.com
Report problems or general comments to Sanjeev Gupta, Tom Johndrew.
TrainTracker provides five query flavors
By Bug ID Click here for detailed help
User can enter a specific DDTS# (e.g , CSCdixxxxx) and receive select information regarding:
what release the defect must be fixed in
what diffs exist
when the fix will be available in both the next interim and next maintenance releases
By Projected bug fix date Click here for detailed help
This query was specially designed with DE managers and CSEs in mind. Given
Netscape
320

FIG. 3

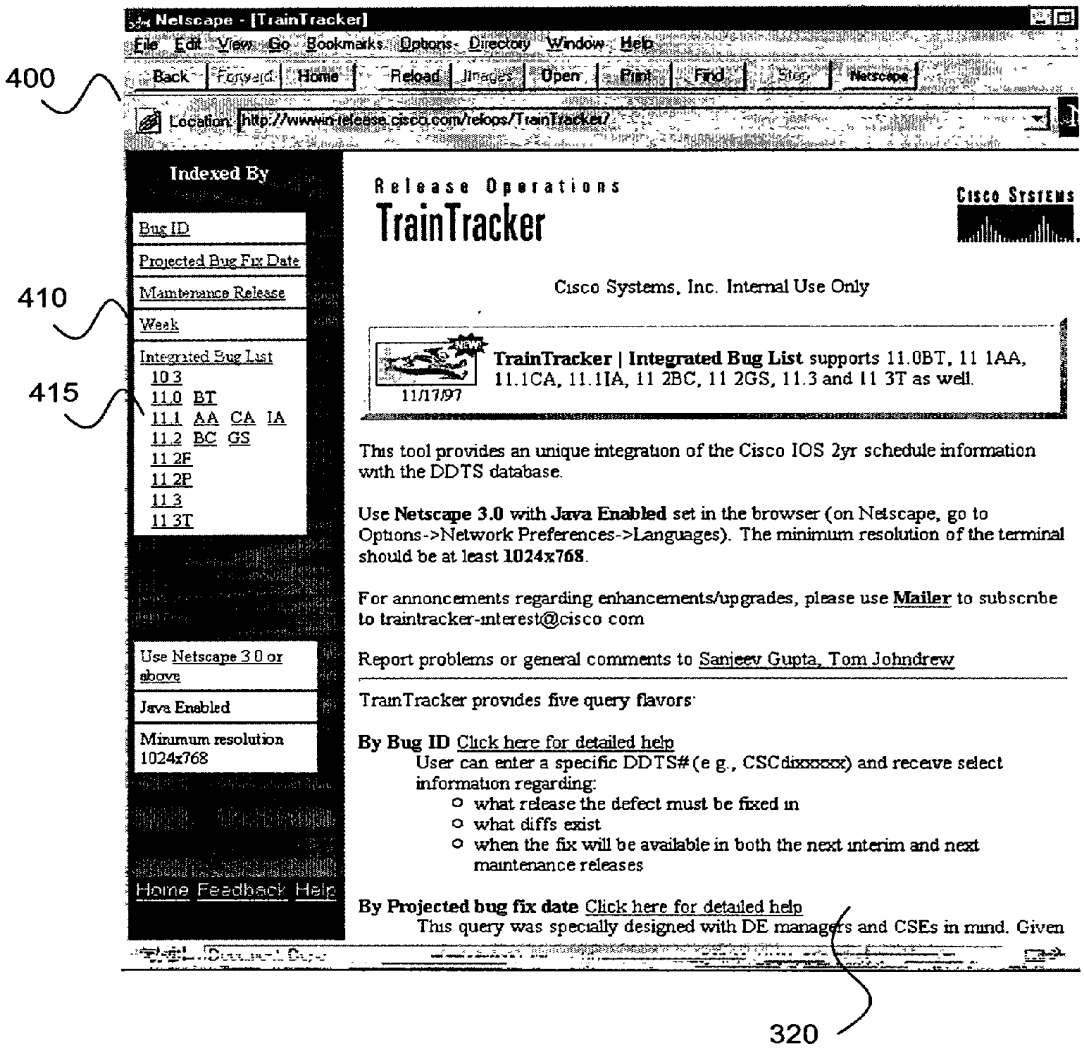

400
410
415
320
Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www-release.cisco.com/relops/TrainTracker/
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
10.3
11.0 BT
11.1 AA CA IA
11.2 BC GS
11.2F
11.2P
11.3
11.3T
Use Netscape 3.0 or above
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Release Operations
TrainTracker
Cisco Systems
Cisco Systems, Inc. Internal Use Only
11/17/97
TrainTracker | Integrated Bug List supports 11.0BT, 11.1AA, 11.1CA, 11.1IA, 11.2BC, 11.2GS, 11.3 and 11.3T as well.
This tool provides an unique integration of the Cisco IOS 2yr schedule information with the DDTS database.
Use Netscape 3.0 with Java Enabled set in the browser (on Netscape, go to Options->Network Preferences->Languages). The minimum resolution of the terminal should be at least 1024x768.
For announcements regarding enhancements/upgrades, please use Mailer to subscribe to traintracker-interest@cisco.com
Report problems or general comments to Sanjeev Gupta, Tom Johndrew
TrainTracker provides five query flavors:
By Bug ID Click here for detailed help
User can enter a specific DDTS# (e.g., CSCdixxxxx) and receive select information regarding:
what release the defect must be fixed in
what diffs exist
when the fix will be available in both the next interim and next maintenance releases
By Projected bug fix date Click here for detailed help
This query was specially designed with DE managers and CSEs in mind. Given

FIG. 4

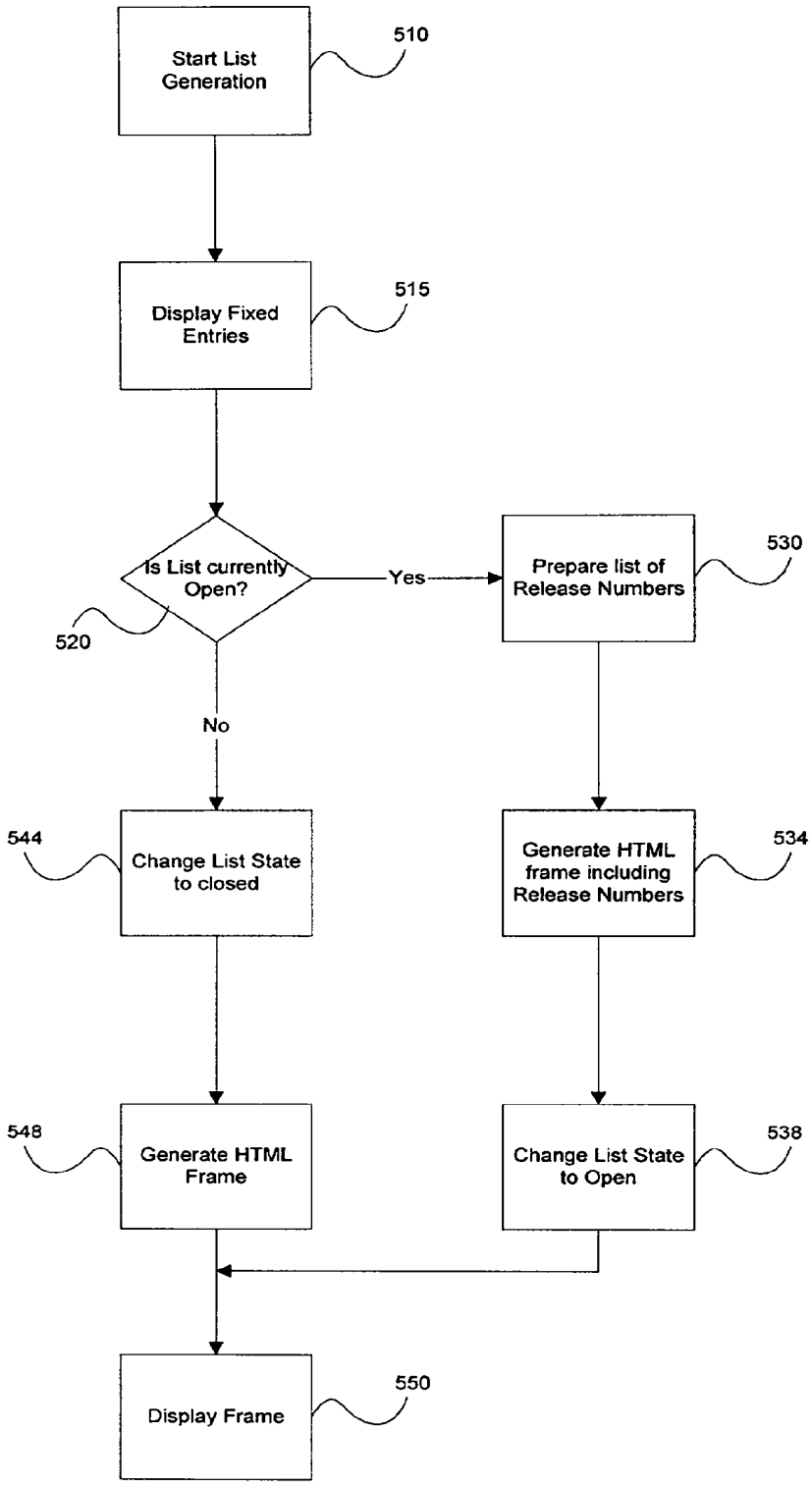

Start List Generation
510
Display Fixed Entries
515
Is List currently Open?
520
Yes
Prepare list of Release Numbers
530
No
544
Change List State to closed
Generate HTML frame including Release Numbers
534
548
Generate HTML Frame
Change List State to Open
538
Display Frame
550

FIG. 5

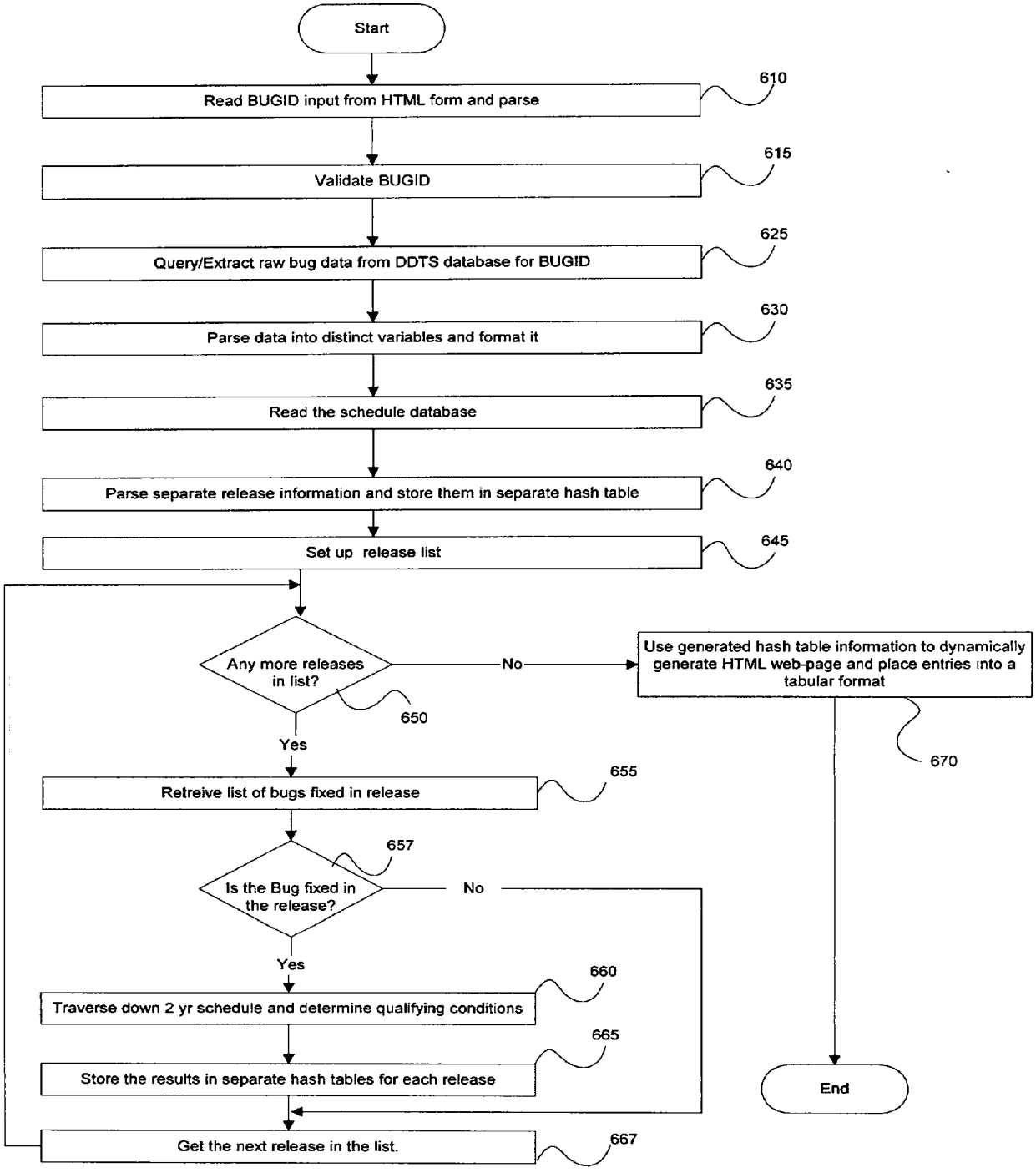

Start
Read BUGID input from HTML form and parse
610
Validate BUGID
615
Query/Extract raw bug data from DDTS database for BUGID
625
Parse data into distinct variables and format it
630
Read the schedule database
635
Parse separate release information and store them in separate hash table
640
Set up release list
645
Any more releases in list?
No
650
Yes
Use generated hash table information to dynamically generate HTML web-page and place entries into a tabular format
670
Retreive list of bugs fixed in release
655
Is the Bug fixed in the release?
657
No
Yes
Traverse down 2 yr schedule and determine qualifying conditions
660
Store the results in separate hash tables for each release
665
Get the next release in the list.
667
End

FIG. 6

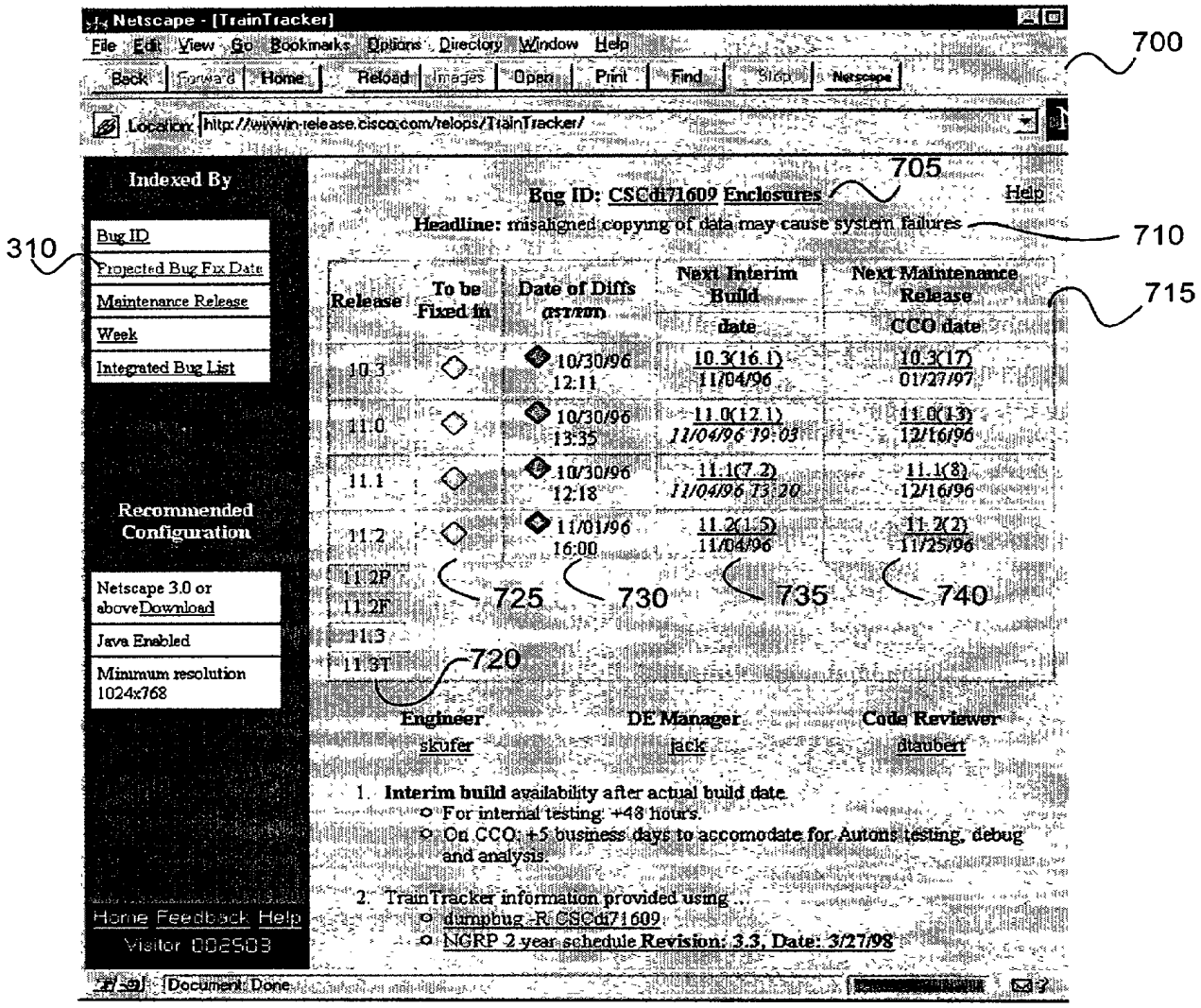

Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www-release.cisco.com/relops/TrainTracker/
700
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
310
Recommended Configuration
Netscape 3.0 or above Download
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 002503
Bug ID: CSCdi71609 Enclosures
705
Help
Headline: misaligned copying of data may cause system failures
710
Release
To be Fixed in
Date of Diffs gstrun
Next Interim Build date
Next Maintenance Release CCO date
715
10.3
10/30/96 12:11
10.3(16.1) 11/04/96
10.3(17) 01/27/97
11.0
10/30/96 13:35
11.0(12.1) 11/04/96 19:03
11.0(13) 12/16/96
11.1
10/30/96 12:18
11.1(7.2) 11/04/96 13:20
11.1(8) 12/16/96
11.2
11/01/96 16:00
11.2(1.5) 11/04/96
11.2(2) 11/25/96
11.2P
11.2F
11.3
11.3T
720
725
730
735
740
Engineer skufer
DE Manager jack
Code Reviewer daubert
1. Interim build availability after actual build date.
For internal testing: +48 hours.
On CCO: +5 business days to accomodate for Autons testing, debug and analysis.
2. TrainTracker information provided using...
dumpbug -R CSCdi71609
NGRP 2 year schedule Revision: 3.3, Date: 3/27/98
Document Done

FIG. 7

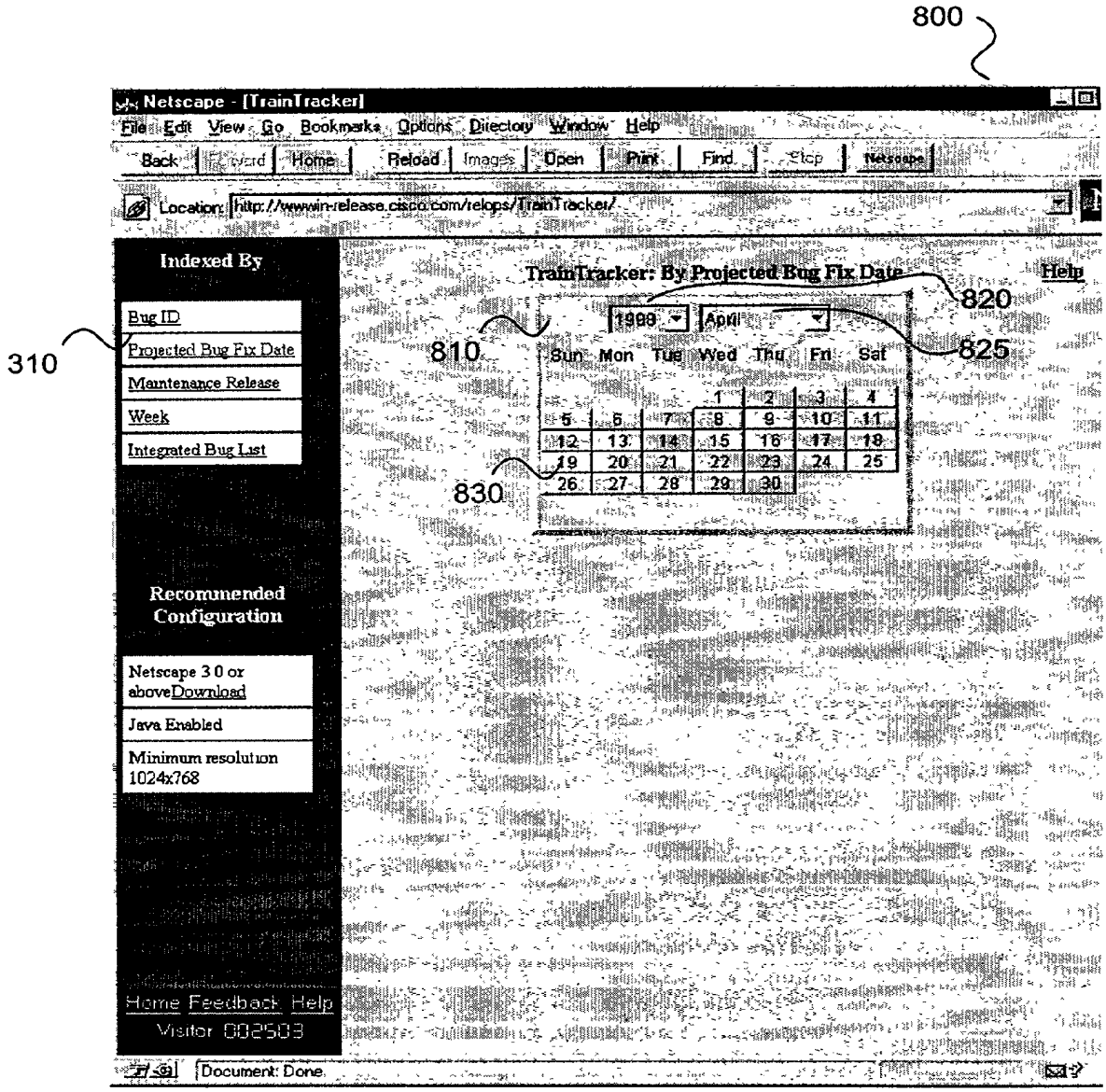

800
Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://wwwin-release.cisco.com/relops/TrainTracker/
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
310
Recommended Configuration
Netscape 3.0 or aboveDownload
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 002503
TrainTracker: By Projected Bug Fix Date
Help
820
1999
April
810
825
Sun Mon Tue Wed Thu Fri Sat
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
830
Document: Done

FIG. 8

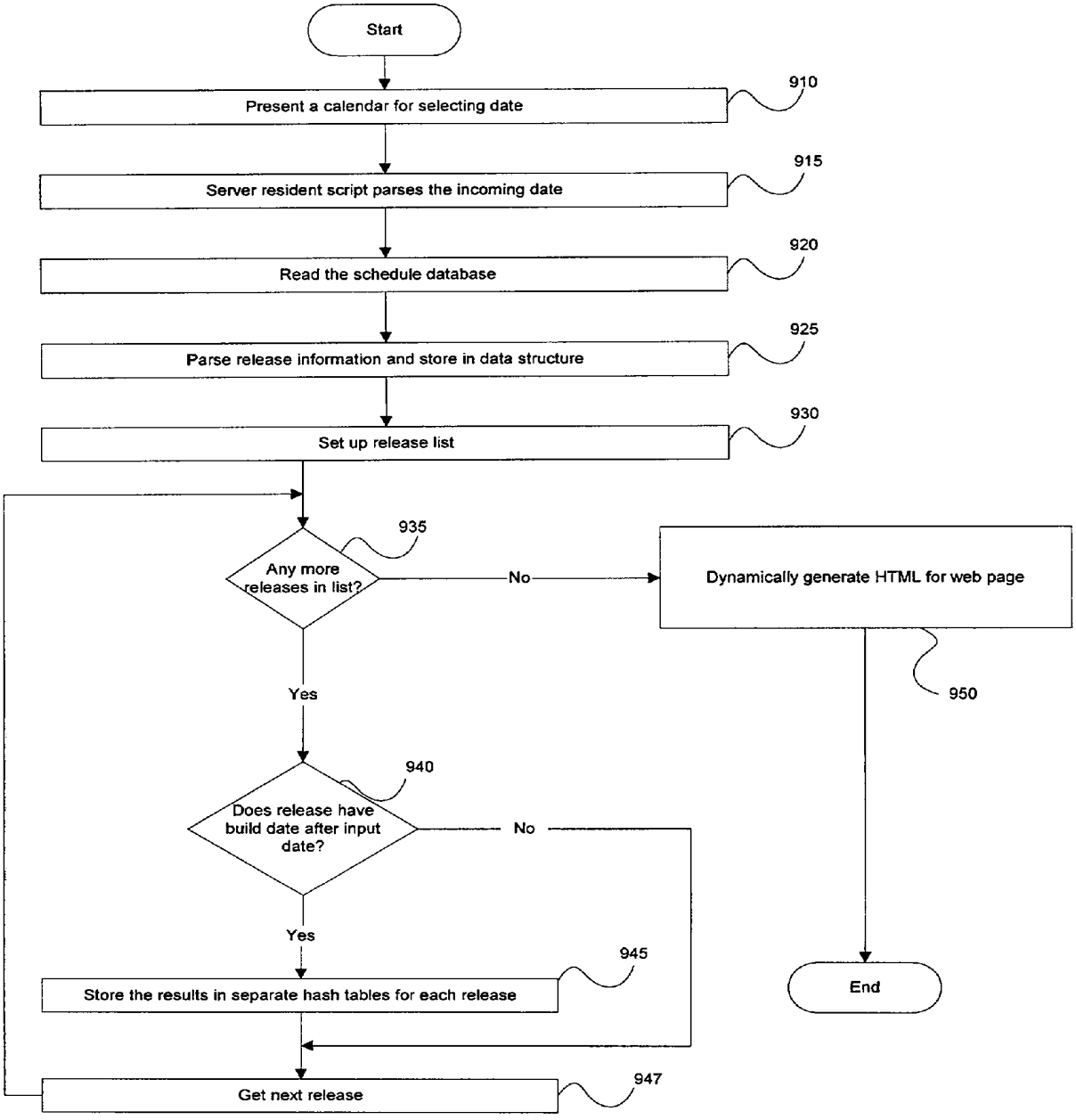

Start
Present a calendar for selecting date
910
Server resident script parses the incoming date
915
Read the schedule database
920
Parse release information and store in data structure
925
Set up release list
930
Any more releases in list?
935
No
Dynamically generate HTML for web page
950
Yes
Does release have build date after input date?
940
No
Yes
Store the results in separate hash tables for each release
945
End
Get next release
947

FIG. 9

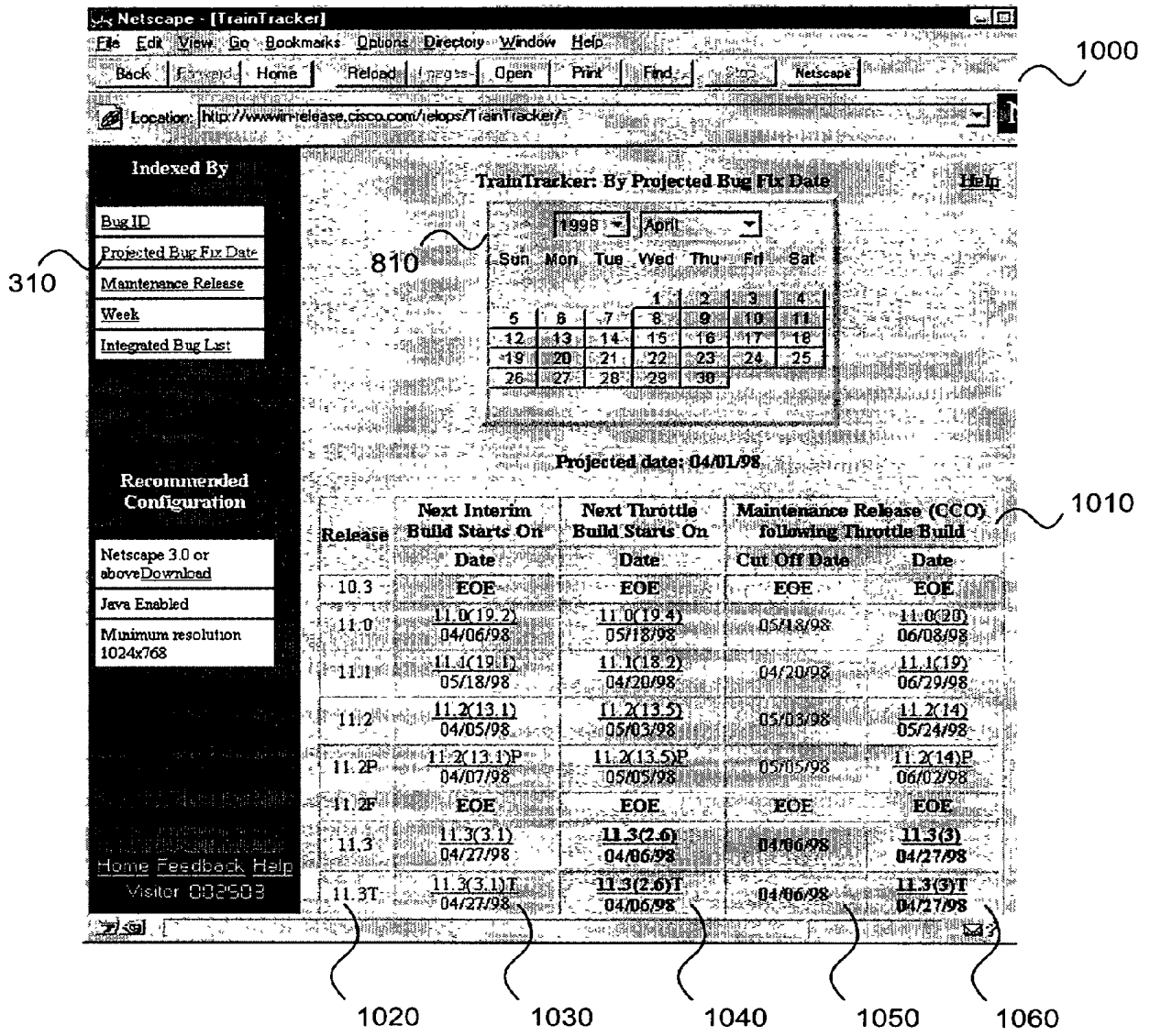

Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www.rr-release.cisco.com/relops/TrainTracker/
1000
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
310
TrainTracker: By Projected Bug Fix Date
Help
1998 April
Sun Mon Tue Wed Thu Fri Sat
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
810
Projected date: 04/01/98
Recommended Configuration
Netscape 3.0 or aboveDownload
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 002503
1010
Release
Next Interim Build Starts On Date
Next Throttle Build Starts On Date
Maintenance Release (CCO) following Throttle Build Cut Off Date
Maintenance Release (CCO) following Throttle Build Date
10.3 EOE EOE EOE EOE
11.0 11.0(19.2) 04/06/98 11.0(19.4) 05/18/98 05/18/98 11.0(20) 06/08/98
11.1 11.1(19.1) 05/18/98 11.1(18.2) 04/20/98 04/20/98 11.1(19) 06/29/98
11.2 11.2(13.1) 04/05/98 11.2(13.5) 05/03/98 05/03/98 11.2(14) 05/24/98
11.2P 11.2(13.1)P 04/07/98 11.2(13.5)P 05/05/98 05/05/98 11.2(14)P 06/02/98
11.2F EOE EOE EOE EOE
11.3 11.3(3.1) 04/27/98 11.3(2.6) 04/06/98 04/06/98 11.3(3) 04/27/98
11.3T 11.3(3.1)T 04/27/98 11.3(2.6)T 04/06/98 04/06/98 11.3(3)T 04/27/98
1020
1030
1040
1050
1060

FIG. 10

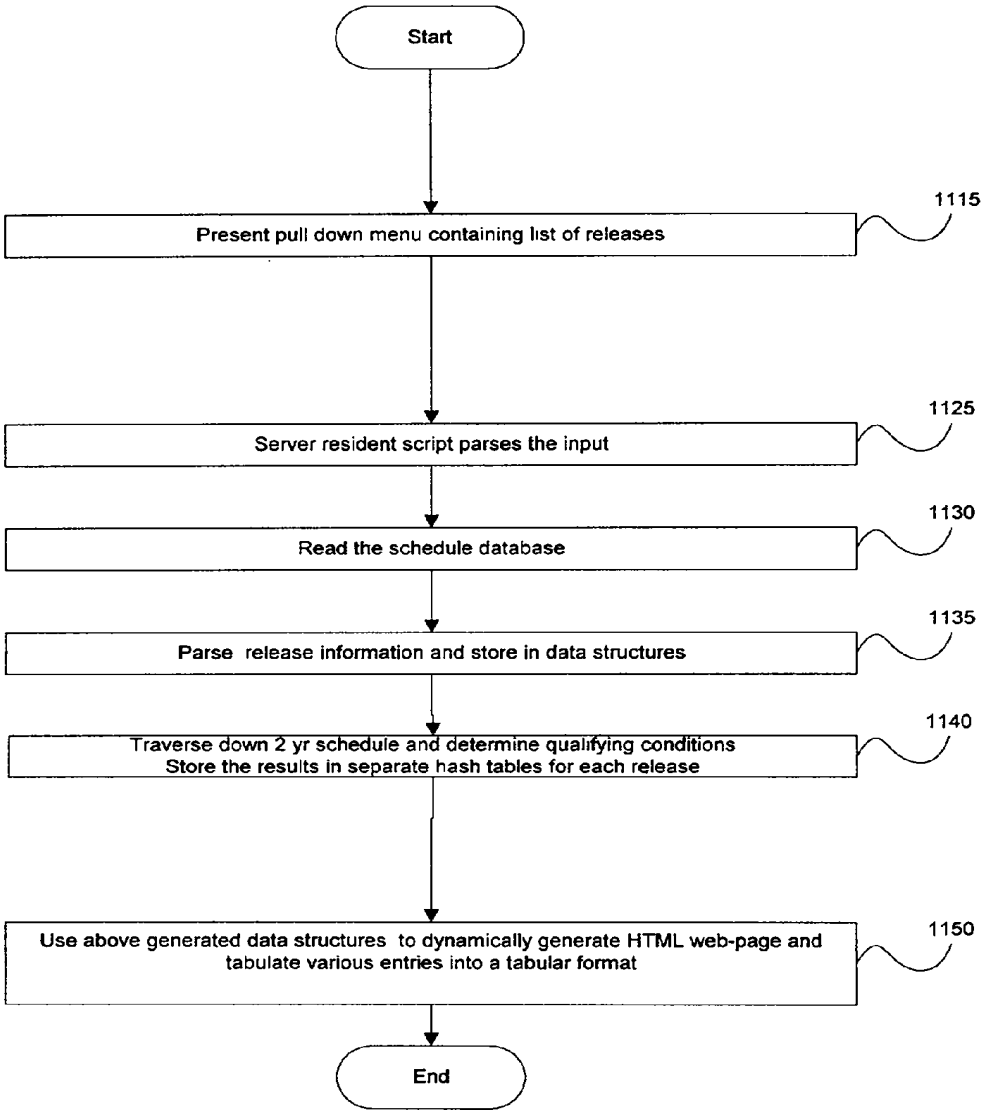

Start
1115
Present pull down menu containing list of releases
1125
Server resident script parses the input
1130
Read the schedule database
1135
Parse release information and store in data structures
1140
Traverse down 2 yr schedule and determine qualifying conditions
Store the results in separate hash tables for each release
1150
Use above generated data structures to dynamically generate HTML web-page and tabulate various entries into a tabular format
End

FIG. 11

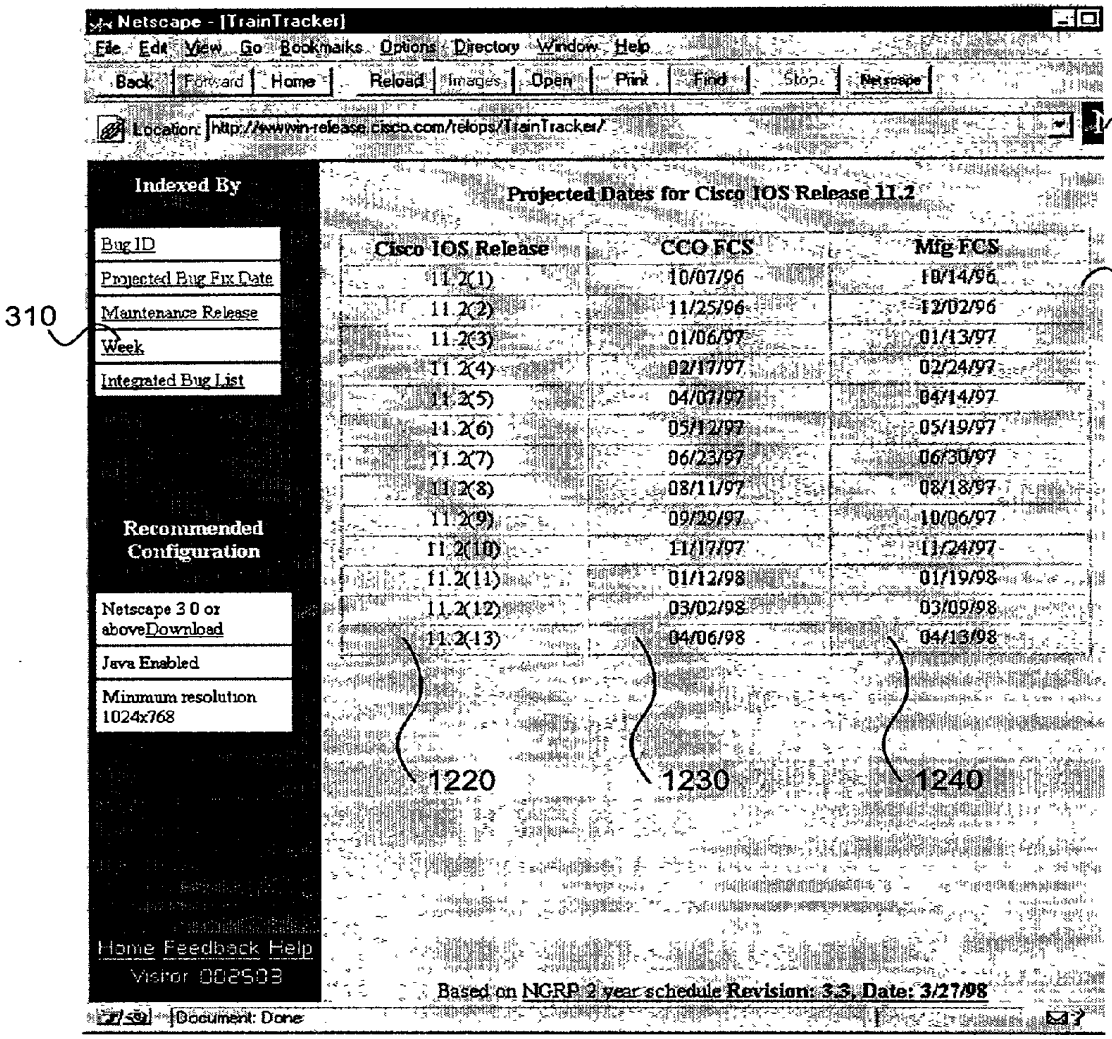

Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://wwwin-release.cisco.com/relops/TrainTracker/
1200
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
310
Recommended Configuration
Netscape 3.0 or aboveDownload
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 0026503
Projected Dates for Cisco IOS Release 11.2
1210
Cisco IOS Release | CCO FCS | Mfg FCS
11.2(1) | 10/07/96 | 10/14/96
11.2(2) | 11/25/96 | 12/02/96
11.2(3) | 01/06/97 | 01/13/97
11.2(4) | 02/17/97 | 02/24/97
11.2(5) | 04/07/97 | 04/14/97
11.2(6) | 05/12/97 | 05/19/97
11.2(7) | 06/23/97 | 06/30/97
11.2(8) | 08/11/97 | 08/18/97
11.2(9) | 09/29/97 | 10/06/97
11.2(10) | 11/17/97 | 11/24/97
11.2(11) | 01/12/98 | 01/19/98
11.2(12) | 03/02/98 | 03/09/98
11.2(13) | 04/06/98 | 04/13/98
1220
1230
1240
Based on NGRP 2 year schedule Revision: 3.3, Date: 3/27/98
Document: Done

FIG. 12

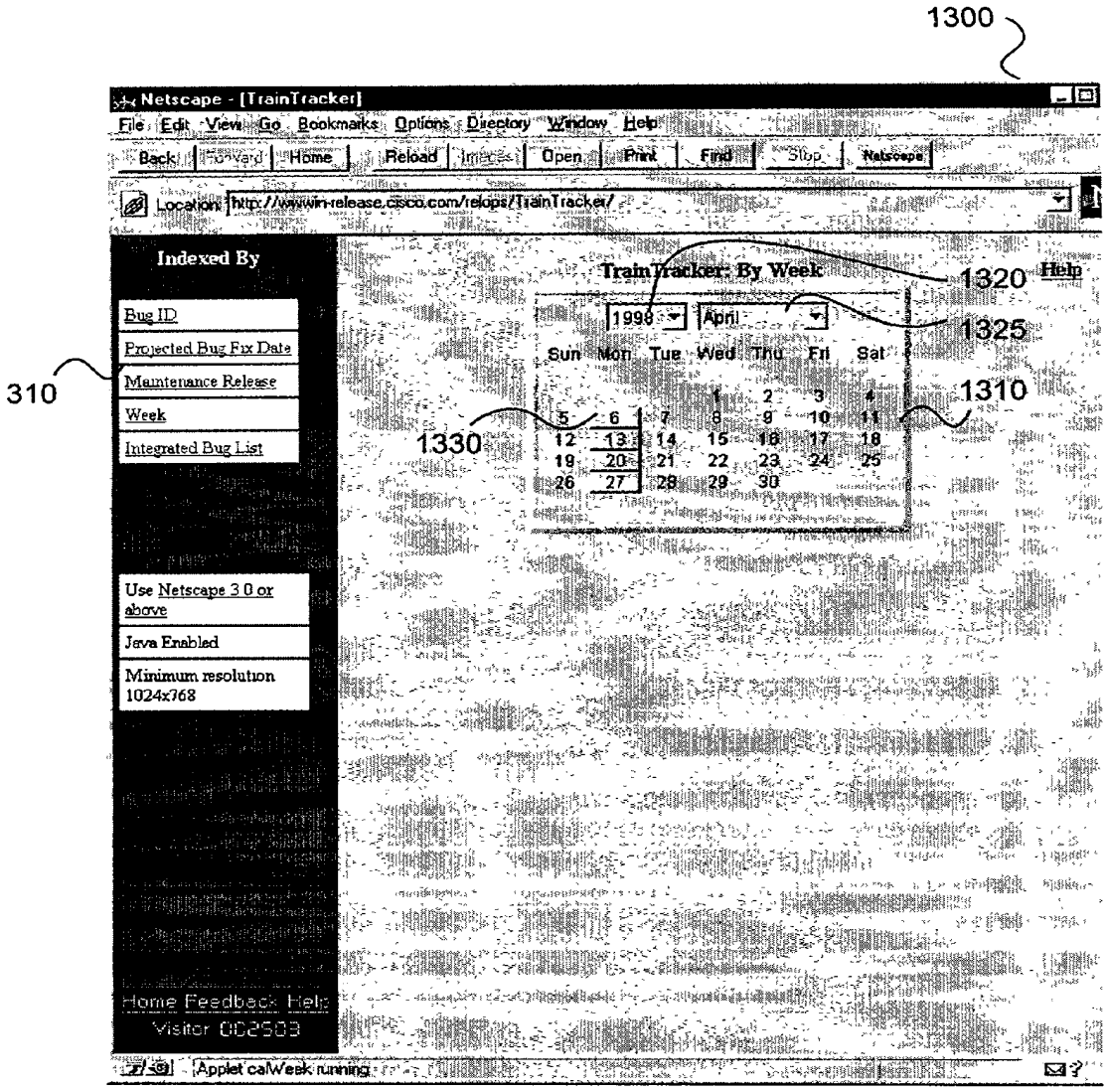

1300
Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www.in-release.cisco.com/relops/TrainTracker/
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
310
Use Netscape 3.0 or above
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 002503
TrainTracker: By Week
1320
Help
1998
April
1325
Sun Mon Tue Wed Thu Fri Sat
1330
1310
Applet calWeek running

FIG. 13

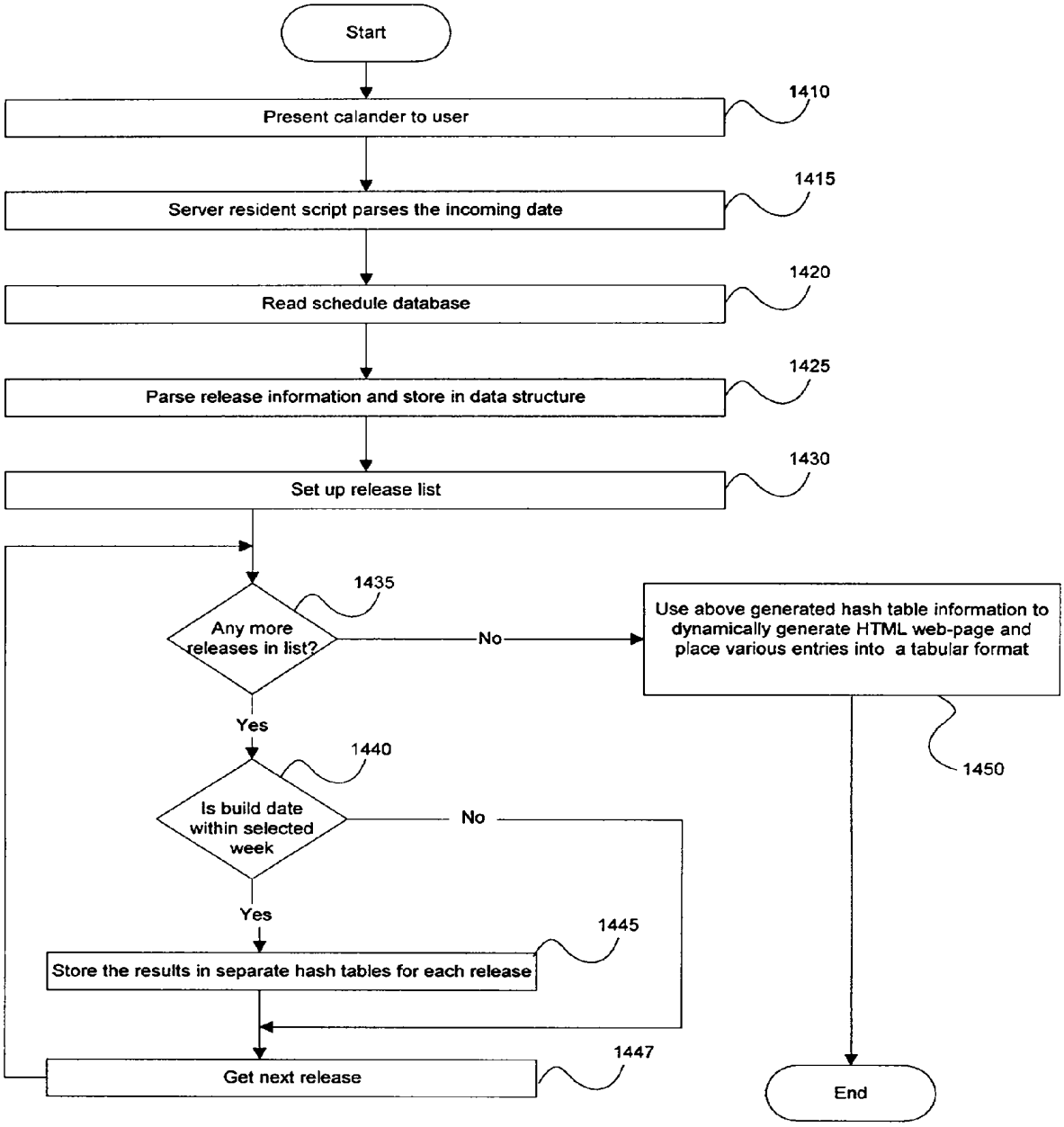

Start
Present calander to user
1410
Server resident script parses the incoming date
1415
Read schedule database
1420
Parse release information and store in data structure
1425
Set up release list
1430
Any more releases in list?
1435
No
Use above generated hash table information to dynamically generate HTML web-page and place various entries into a tabular format
1450
Yes
Is build date within selected week
1440
No
Yes
Store the results in separate hash tables for each release
1445
Get next release
1447
End

FIG. 14

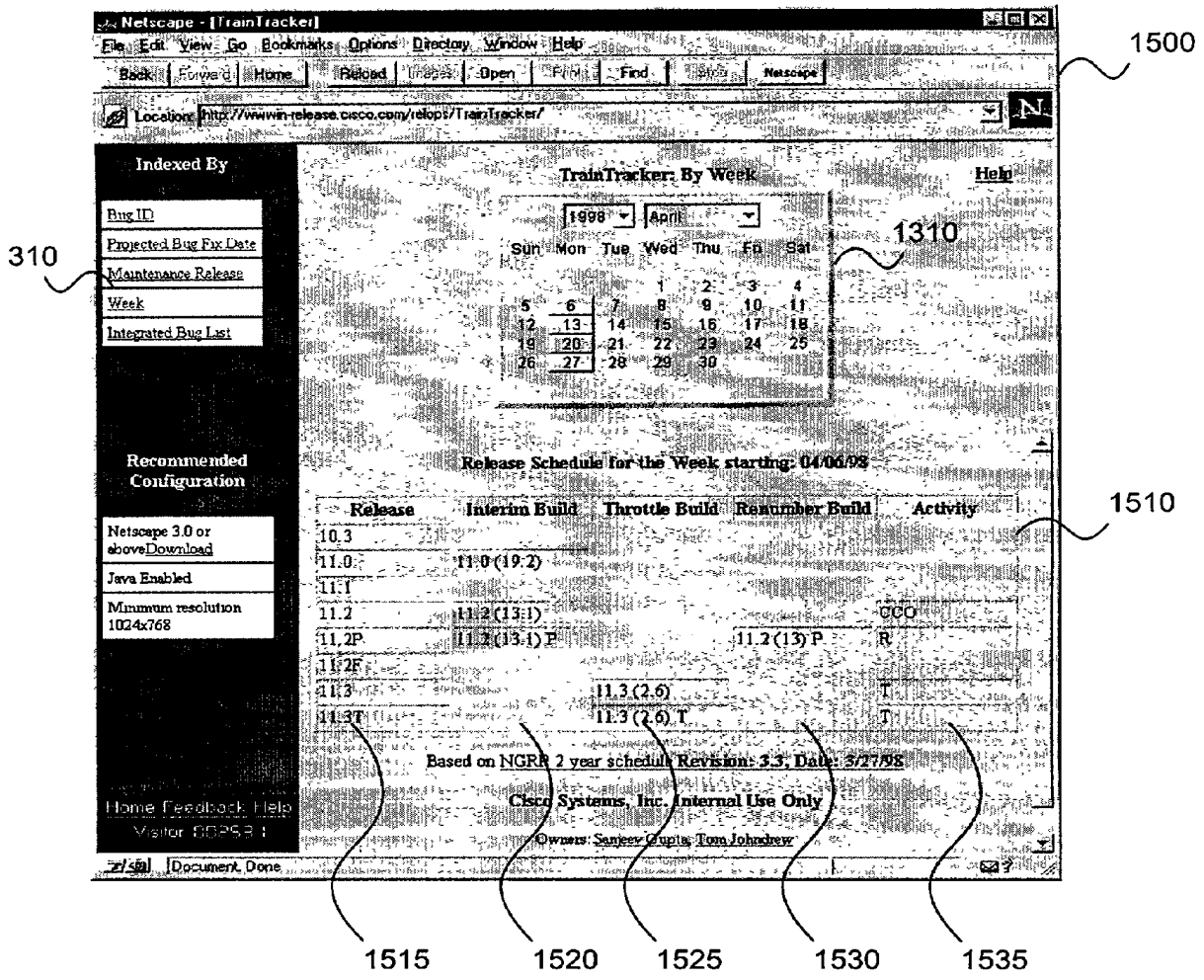

1500
310
1310
1510
1515
1520
1525
1530
1535
Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www-release.cisco.com/relops/TrainTracker/
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
Recommended Configuration
Netscape 3.0 or aboveDownload
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 88958 !
TrainTracker: By Week
Help
1998
April
Sun Mon Tue Wed Thu Fri Sat
Release Schedule for the Week starting: 04/06/98
Release
Interim Build
Throttle Build
Renumber Build
Activity
10.3
11.0
11.0 (19.2)
11.1
11.2
11.2 (13.1)
CCO
11.2P
11.2 (13.1) P
11.2 (13) P
R
11.2F
11.3
11.3 (2.6)
T
11.3T
11.3 (2.6) T
T
Based on NGRR 2 year schedule Revision: 3.3, Date: 3/27/98
Cisco Systems, Inc. Internal Use Only
Owners: Sanjeev Gupta, Tom Johnsrew
Document Done

FIG. 15

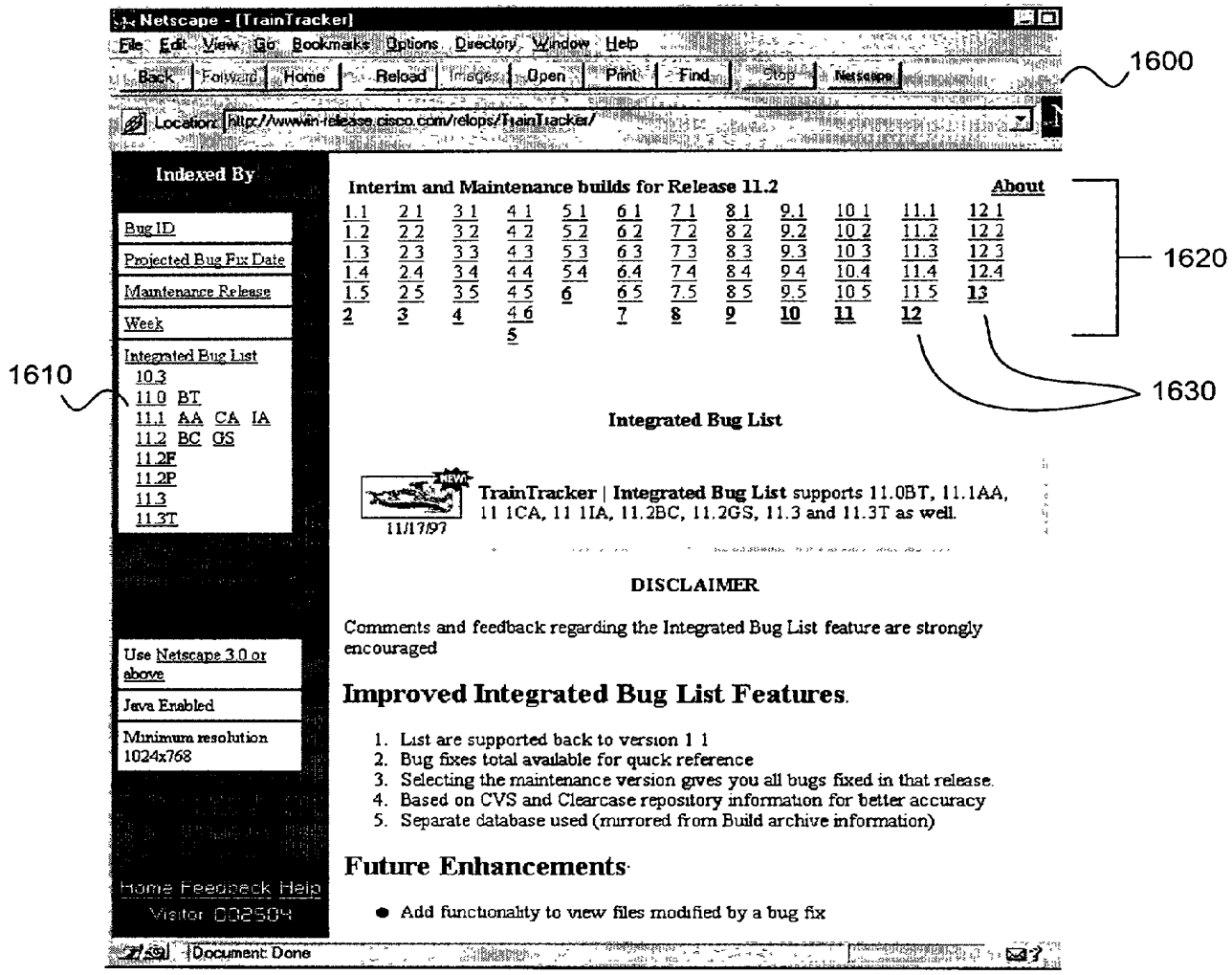

Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www-in-release.cisco.com/relops/TrainTracker/
1600
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
10.3
11.0 BT
11.1 AA CA IA
11.2 BC GS
11.2F
11.2P
11.3
11.3T
1610
Use Netscape 3.0 or above
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 002504
Interim and Maintenance builds for Release 11.2
About
1.1 2.1 3.1 4.1 5.1 6.1 7.1 8.1 9.1 10.1 11.1 12.1
1.2 2.2 3.2 4.2 5.2 6.2 7.2 8.2 9.2 10.2 11.2 12.2
1.3 2.3 3.3 4.3 5.3 6.3 7.3 8.3 9.3 10.3 11.3 12.3
1.4 2.4 3.4 4.4 5.4 6.4 7.4 8.4 9.4 10.4 11.4 12.4
1.5 2.5 3.5 4.5 6 6.5 7.5 8.5 9.5 10.5 11.5 13
2 3 4 4.6 5 7 8 9 10 11 12
1620
1630
Integrated Bug List
11/17/97
TrainTracker | Integrated Bug List supports 11.0BT, 11.1AA, 11.1CA, 11.1IA, 11.2BC, 11.2GS, 11.3 and 11.3T as well.
DISCLAIMER
Comments and feedback regarding the Integrated Bug List feature are strongly encouraged
Improved Integrated Bug List Features.
1. List are supported back to version 1.1
2. Bug fixes total available for quick reference
3. Selecting the maintenance version gives you all bugs fixed in that release.
4. Based on CVS and Clearcase repository information for better accuracy
5. Separate database used (mirrored from Build archive information)
Future Enhancements.
Add functionality to view files modified by a bug fix
Document: Done

FIG. 16

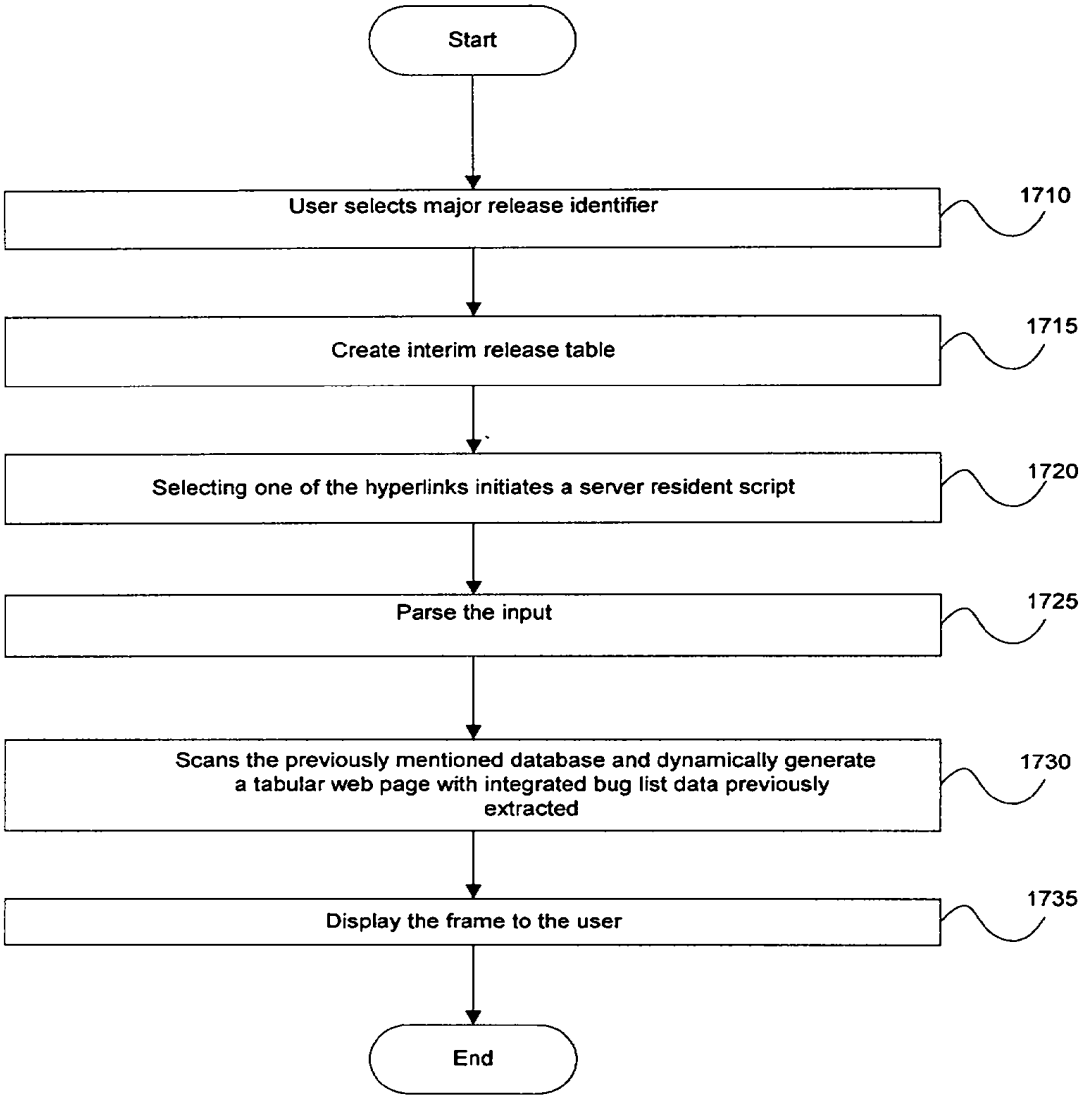

Start
User selects major release identifier
1710
Create interim release table
1715
Selecting one of the hyperlinks initiates a server resident script
1720
Parse the input
1725
Scans the previously mentioned database and dynamically generate a tabular web page with integrated bug list data previously extracted
1730
Display the frame to the user
1735
End

FIG. 17

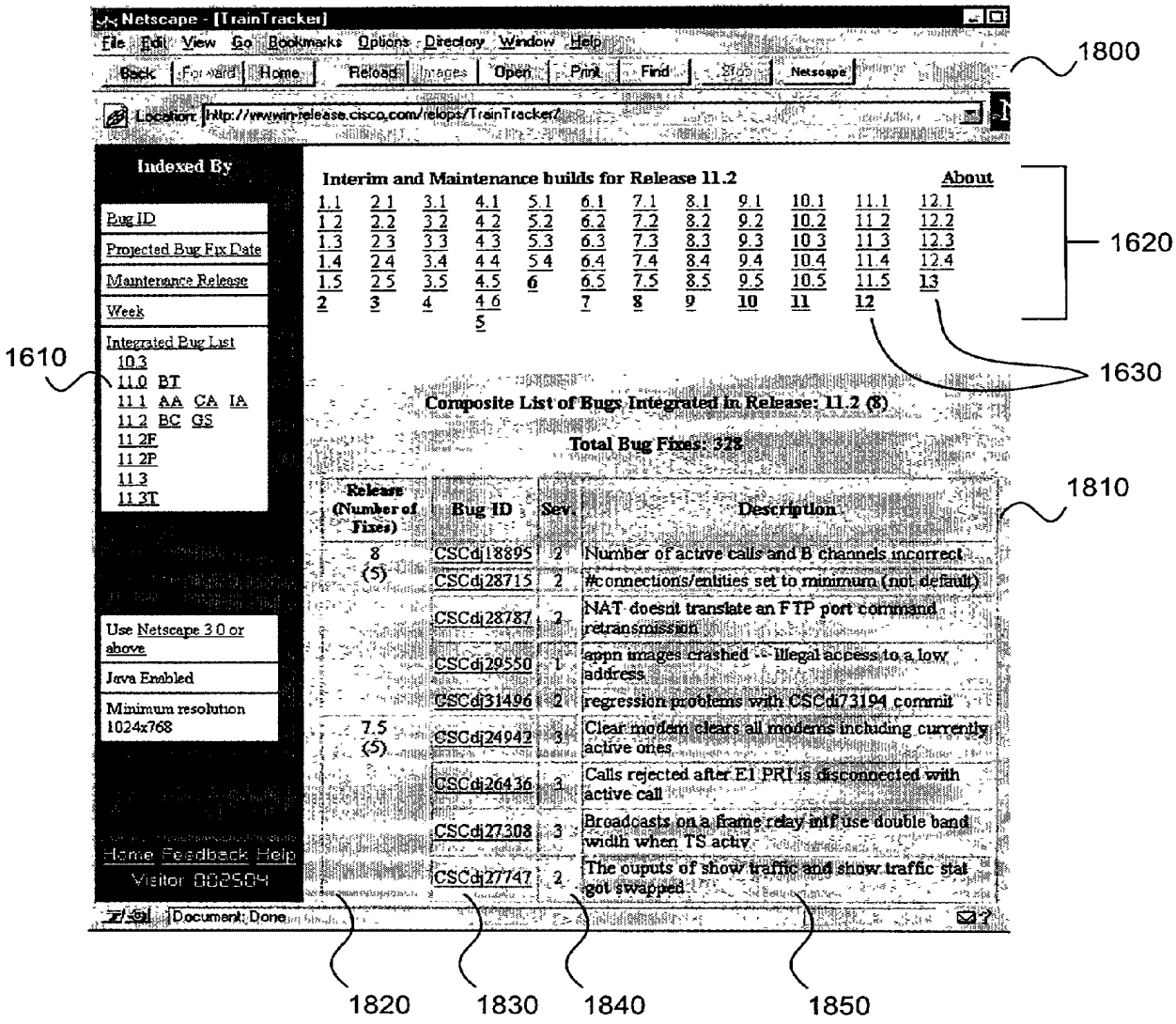

Netscape - [TrainTracker]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop Netscape
Location: http://www-release.cisco.com/relops/TrainTracker/
1800
Indexed By
Bug ID
Projected Bug Fix Date
Maintenance Release
Week
Integrated Bug List
10.3
11.0 BT
11.1 AA CA IA
11.2 BC GS
11.2F
11.2P
11.3
11.3T
Use Netscape 3.0 or above
Java Enabled
Minimum resolution 1024x768
Home Feedback Help
Visitor 882584
Interim and Maintenance builds for Release 11.2
About
1.1 1.2 1.3 1.4 1.5 2
2.1 2.2 2.3 2.4 2.5 3
3.1 3.2 3.3 3.4 3.5 4
4.1 4.2 4.3 4.4 4.5 4.6 5
5.1 5.2 5.3 5.4 6
6.1 6.2 6.3 6.4 6.5 7
7.1 7.2 7.3 7.4 7.5 8
8.1 8.2 8.3 8.4 8.5 9
9.1 9.2 9.3 9.4 9.5 10
10.1 10.2 10.3 10.4 10.5 11
11.1 11.2 11.3 11.4 11.5 12
12.1 12.2 12.3 12.4 13
1620
1610
1630
Composite List of Bugs Integrated in Release: 11.2 (8)
Total Bug Fixes: 328
1810
Release (Number of Fixes)
Bug ID
Sev.
Description
8 (5)
CSCdj18895 2 Number of active calls and B channels incorrect
CSCdj28715 2 #connections/entities set to minimum (not default)
CSCdj28787 2 NAT doesnt translate an FTP port command retransmission
CSCdj29550 1 appn images crashed -- illegal access to a low address
CSCdj31496 2 regression problems with CSCdj73194 commit
7.5 (5)
CSCdj24942 3 Clear modem clears all modems including currently active ones
CSCdj26436 3 Calls rejected after E1 PRI is disconnected with active call
CSCdj27308 3 Broadcasts on a frame relay intf use double band width when TS activ
CSCdj27747 2 The ouputs of show traffic and show traffic stat got swapped
Document: Done
1820
1830
1840
1850

FIG. 18

SYSTEM AND METHOD FOR RETRIEVING SOFTWARE RELEASE INFORMATION

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] The present invention relates generally to methods for obtaining information about software, and more particularly for obtaining and displaying information about software releases.

[0003] 2. Discussion of Background Art

[0004] The development and maintenance of computer software is affected by a number of issues. There are an almost innumerable number of combinations of hardware, software and network environments that must be supported. Often, many of these environments are incompatible with each other. In addition, each of the environments undergoes rapid change as technology advances in their respective areas. As a result, it can be expected that a computer software product will require many changes during its life cycle.

[0005] A typical software development process has four major phases. The first is the design phase, where the operating characteristics of the software are defined. This is followed by the development phase, where programmers and engineers write the software to meet the requirements defined during the design phase. The third phase is the test phase, where the software is tested to ensure that it meets the requirements defined during the design phase, and has no major problems. The fourth phase is the release phase, where the software is provided to customers and other end users. During the development, test and release phases of the software life cycle, problems or design issues may be encountered where the software does not operate in a desirable way. These problems are often called "bugs." In addition, it is often desirable to add new features to the software as customer needs become more evident. Fixing bugs and adding features require the initiation of a new cycle of design, development, test and release phases.

[0006] It is typical in the software industry to organize releases of new software in a hierarchical manner. A major release of software is one in which significant changes have been made to the software. For example, significant new functionality may have been added or removed, or the software's user interface may have been dramatically changed. A minor release is typically one in which bugs have been fixed and improvements have been added to the software, but no major functionality has changed. In addition, there may be other releases of software that are below the level of a minor release, such as a maintenance release, an interim release, a test release or an emergency bug-fix release. These releases typically do not encompass as many bug fixes or other changes to the software. All of these releases are commonly assigned identifiers in order to distinguish them from one another. A commonly used format for identifiers is a series of numbers, separated by periods or enclosed within parentheses which identify the major and maintenance release levels for the software. For example, the identifier "11.2(13)" may indicate major release 11.2 and maintenance release 13 within that major release.

[0007] In addition, build numbers may be used in combination with release numbers to identify software. A software build is the process of compiling and linking all of the components of a software product. Typically many different programs and modules make up a software product, and they must all be assembled into a package. A build number is essentially a sequence number assigned to the software. For example an identifier "11.2(6.1)" may indicate build 6.1 for major release 11.2 of a software product. During the development and maintenance process many reasons for building software occur. For example a maintenance build is a software build intended to be released to customers that fixes bugs that have been discovered since the last release of the software. An interim build is intended to be used by internal users to test the software before releasing it to customers or for repairing software in response to urgent customer problems. A throttle build is used for final testing of a software product before final approval, and therefore must incorporate very tight change controls. A renumber build is one that has been tested and is built again to be labeled with a new release number or identifier.

[0008] Many different parties with varying needs are acutely interested in the status of particular releases of software. Software developers need to know when the software they are working on must be ready in order to meet the deadline for a scheduled release. Customers want to know when software will be available that fixes problems they are encountering. Technical writers want to know what bug fixes and features will be included in new releases so that they can create documentation on the bug-fix or features. Product managers and customer service engineers want to know what bugs or features are included in a release so that they may accurately convey the information to current or potential customers. Each of these parties has a wide range of expertise in dealing with computers and software, varying from novice to expert.

[0009] In addition, these parties may be in very different locations. The development staff may be in one location, the maintenance staff in another, and the technical writers in yet another location. Customer service engineers and the customers may be scattered around the world.

[0010] In the past, specialized products have been developed to meet some of the individual needs of these parties. Software source code control systems have been created to allow software developers to control and identify changes to software. Additionally, software developers have benefited from software development environments which have been created to control the compiling and linking that occurs during the build process. Other systems and methods have been created to control and maintain the schedule of major and minor releases. Still other systems have been developed to report and track the history of bugs and enhancement requests. Each of these systems typically maintains their own database, each has their own user interface, and each may be centrally located and controlled.

[0011] Unfortunately, for a person to obtain complete and accurate information on a software release, he or she must have access to and knowledge of how to use the various tools that have been developed to address the individual aspects of software development issues detailed above. In addition, the user must often either have the required software on their own personal computer or be located on the same internal corporate network.

[0012] In response to these concerns, what is needed is a method for retrieving software release information that can

be used in a variety of locations and environments, that can be used by novices and experts alike, and that overcomes the problems of the prior art.

SUMMARY OF THE INVENTION

[0013] The present invention provides a system and method for retrieving software release information. One component within the system of the present invention automatically gathers data from various databases and displays information on the releases containing a fix for an identified defect. A second component determines a date by which a bug must fixed if it is to be included in an identified release. A third component presents the dates of various software release events associated with a particular release. A fourth component identifies all of the software release events that are to occur within a user specified week. A fifth component automatically generates a list of defects already integrated into a particular software release.

[0014] All of the above described components dynamically generate HTML defining a web page thereby enabling the display of the software release information on displays located anywhere in the world having a network connection to a computer implementing the system and method of the present invention.

[0015] These and other aspects of the invention will be recognized by those skilled in the art upon review of the detailed description, drawings, and claims set forth below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] **FIG. 1** is a block diagram of a computer system including a software release information system in accordance with the present invention;

[0017] **FIG. 2** is a block diagram of the software release information system;

[0018] **FIG. 3** is a graphical depiction of the initial display output of the preferred embodiment;

[0019] **FIG. 4** is a graphical depiction of the display output after the "Integrated Bug List" link in **FIG. 3** has been selected;

[0020] **FIG. 5** is a flowchart describing the transition from the display output in **FIG. 3** to the display output in **FIG. 4**;

[0021] **FIG. 6** is a flowchart for collecting data based on Bug ID;

[0022] **FIG. 7** is a graphical depiction of a display output of the data collection produced using the process of **FIG. 6**;

[0023] **FIG. 8** is graphical depiction of a display output allowing a user to select a particular Projected Bug Fix Date;

[0024] **FIG. 9** is a flowchart for collecting data based on a Projected Bug Fix Date;

[0025] **FIG. 10** is a graphical depiction of a display output of the data collection produced using the process of **FIG. 9**;

[0026] **FIG. 11** is a flowchart for collecting data based on a Maintenance Release Identifier;

[0027] **FIG. 12** is a graphical depiction of a display output of the data collection produced using the process of **FIG. 11**;

[0028] **FIG. 13** is graphical depiction of a display output allowing a user to select a particular week of a year ;

[0029] **FIG. 14** is a flowchart for collecting data based on an input of a particular week of a year;

[0030] **FIG. 15** is a graphical depiction of a display output showing the data collection produced using the process described in **FIG. 14**;

[0031] **FIG. 16** is a graphical depiction of a display output for allowing the selection of a particular release identifier;

[0032] **FIG. 17** is a flowchart for gathering a data collection based on the selection of a particular release identifier; and

[0033] **FIG. 18** is a graphical depiction if a display output showing the data collection produced using the process described in **FIG. 17**.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0034] **FIG. 1** is a block diagram of a computer system **100**, which includes one or more Central Processing Units **110**, an input device **115** such as a keyboard or a mouse, an output device **120** such as a Cathode Ray Tube (CRT) display, Random Access Memory (RAM) **125**, data storage **150** including Read Only Memory (ROM) and other long term storage devices such as hard disk drives, CD-ROM Drives or similar devices, and a communications interface **130**, all coupled together via a signal bus **135**. Data storage **150** holds the various databases associated with a software release information retrieval system. Communications interface **130** is connected to a network **140** such as a local area network, a wide area network, or the Internet. Network **140** connects computer system **100** to network clients **145**. Network clients **145** comprise diskless workstations, mainframe systems, minicomputer systems, personal computers, laptop computers, palm computers or television based web browser systems. Network clients **145** can be located anywhere in the world where the network client can connect to network **140**. Additionally, network **140** may connect computer system **100** to remote file and database servers containing databases used with the software release information system.

[0035] **FIG. 2** is a block diagram of the major components of a release information system **200** of the present invention. For interacting with the release information system **200** a preferred user interface is a web browser such as Netscape Navigator or Microsoft's Internet Explorer. Those skilled in the art will recognize that while the use of such browsers allows the release information system **200** to display information in a manner independent of device and hardware platform, there exist alternatives for displaying the information produced by release information system **200**, which could be readily substituted. The components of the invention are implemented preferably using the Java or Perl programming languages, however other programming languages could be substituted.

[0036] In release information system **200** the first significant display output is the Main Selection Interface **210** which lists the major sub-components of system **200** and allows a user to select and invoke a desired sub-component. Preferably, the display output list of major sub-components

is a list of hyperlinks which upon selection take the user to the selected sub-component interface. The present invention includes sub-component modules to present information to the user by Bug ID **215**, Projected Bug Fix Date **220**, Maintenance Release Identifier **225**, Week **230** and Integrated Bug List **235**.

[**0037**] **FIG. 3** shows the initial display output of the preferred embodiment. Initial screen **300** is a page of output data and is shown as it would appear when displayed using the web browser Netscape Navigator. Initial screen **300** comprises two functional areas, a query component selection index **310** and a text region **320**. Query component selection index **310** is preferably implemented as a list of hyperlinks, with each entry of the list representing one of the major components detailed with reference to **FIG. 2**. A user may select a query component by moving a mouse to position a cursor over the desired hyperlink and clicking a mouse button. Some query components when selected may present an expanded list of additional selectable items. In the preferred embodiment, text region **320** displays descriptive text and images about the present invention, and a selection interface to obtain further information about the system.

[**0038**] **FIG. 4** shows the display output after the "Integrated Bug List" link has been selected from query component selection index **310** of **FIG. 3**. Screen **400**, similarly to screen **300**, comprises text region **320** and an expanded query selection index **410** which includes the additional information of a list of selectable major release identifiers **415**.

[**0039**] **FIG. 5** is a flowchart describing the steps involved in the transition from the display output in **FIG. 3** to the display output in **FIG. 4** and back again. (The method is implemented by conventionally available computer equipment shown in **FIG. 1**.) This method is initiated when the user selects the "Integrated Bug List" link from query component selection index **310** in **FIG. 3** or **410** in **FIG. 4**. The method begins in step **510**, with the initialization of conventional data structures. Next, step **515** creates a list of entries common to the display output in **FIG. 3** and **FIG. 4**. In the preferred embodiment, this list comprises hyperlink entries titled "Bug ID,""Projected Bug Fix Date,""Maintenance Release,""Week" and "Integrated Bug List." Next, in step **520**, the method checks a list state variable to see if the list of selectable major release identifiers is currently being displayed (open, as in **FIG. 4**) or not (closed, as in **FIG. 3**).

[**0040**] If the list state variable is currently open, the method proceeds to step **544** and changes the list state variable to closed. Next, in step **548**, an HTML frame is generated to display the list to the user. Those skilled in the art will recognize that alternative methods to HTML are available to generate a screen to be displayed to a user.

[**0041**] If the list state variable is currently set to closed, the method proceeds to step **530** and queries a database to determine a list of currently active release identifiers. The database is comprised of preferably a computer file system directory structure in which directory names indicate releases. Alternative databases such as a file, spreadsheet, relational database management system or object oriented database could be substituted. Next, step **534** generates an HTML frame containing the currently active release from step **530** and the fixed list entries from step **515**. Next, in step **538**, the list state variable is changed to open to reflect the current list state.

[**0042**] In the final step **550** the HTML frame just generated is displayed to the user. This step will display the page in **FIG. 3** if the list state variable is closed or the page in **FIG. 4** if the list state variable is open.

[**0043**] **FIG. 6** is a flowchart for collecting data based on Bug ID and further describes component **210** of **FIG. 2**. The method begins in step **610**, where the user inputs a bug identifier preferably read from an entry on an HTML form or selected from a list of available bug identifiers, entered onto a database form, etc. Next, in step **615**, the bug identifier is compared to a list of valid bug identifiers. If the bug identifier is invalid, no further processing will take place and the user will be allowed to enter an alternative bug identifier. The method proceeds to step **625** where data regarding an identified bug is read from a database containing information about software bugs. Next, in step **630**, a subset of the data returned in step **625** is stored into data structures describing the bug. This data structure is preferably a hash table, with the bug identifier used as a hash key. Alternatives such as an array or linked list could be substituted. Next, in step **635**, a schedule for software releases is read from a software release database. The release schedule database is preferably a tab-delimited file containing a two year span of schedule information. Alternative database mechanisms could be substituted and other time span lengths could be chosen as necessary. The method proceeds to step **640** where relevant data about each release is stored in a data structure describing the software release. This data structure is preferably also a hash table with the release identifier used as a hash key. Next, step **645** builds a data structure representing a list of releases.

[**0044**] Step **650** in **FIG. 6** begins a loop that iterates over each release in the list of releases by checking whether any releases remain in the list. If a release is in the list, the method proceeds to step **655** which retrieves from the release database a list of bugs fixed in each release. In addition, this step adds to the list those bugs that have been committed to be fixed in the release, but have not yet been fixed. Next, step **657** checks whether the current bug identifier is included in the list of bugs actually fixed or committed to be fixed for the release. If not, the method proceeds to step **667**. If the list of bugs includes the current bug identifier, then step **660** retrieves the schedule information for the release. Next, in step **665**, the schedule information from step **660** is stored in a data structure, preferably a hash table, describing the release schedule. Step **667** retrieves the next release in the list. The method then returns to the top of the loop and step **650**.

[**0045**] When no release remains in the list, step **650** directs the method to step **670** in which the information from the data structures describing the bug, and the information in the data structures describing the software release are used to dynamically generate HTML that produces the web page shown in **FIG. 7**. One could substitute other methods of output such as a database form or a PDF file designed to be used by the Adobe Acrobat page file reader.

[**0046**] **FIG. 7** shows the data collected by the process of **FIG. 6**. Bug ID screen **700** includes the query component selection index **310**, a bug identifier header **705**, a bug headline **710** and a release table **715**. Bug identifier header **705** gives the bug identifier associated with the information on the screen ("CSCdi71609" in **FIG. 7**). Bug headline **710**

contains a short one line description of the bug. Release table **715** has rows describing a particular release and columns containing a data element describing an aspect of the release. Data for each column is obtained from the data structures describing the bug, the software release and the release schedule. Column **720** identifies the release. Column **725** indicates the status of the bug with regard to the release, preferably using a colored diamond. A red diamond indicates that the bug is scheduled to be fixed in the release identified in column **720**, but no further information is available on the status of the bug. A green diamond indicates that the bug has been fixed and the software will be integrated into the release identified in column **720**. A cyan diamond indicates that the bug has been fixed, even though it was not scheduled or committed for the release identified in column **720**. A gold diamond indicates that the bug is scheduled to be fixed in the release identified in column **720**. A magenta diamond indicates that a fix for the release has been approved for inclusion into the release identified in column **720**. Column **730** contains the date and time that software fixing the bug was implemented in the release. Column **735** contains the date that an interim build of the release either took place or will take place. Column **740** contains the date that a maintenance release either was released or will be released to customers.

[**0047**] **FIG. 8** shows the initial screen of component **220** of **FIG. 2, a** display output allowing a user to select a particular Projected Bug Fix Date. Input to this component is a date by which a fix for a bug is expected to be implemented. Screen **800** comprises the query component selection index **310** and a calendar **810** that contains a year drop down box **820** and a month drop down box **825**, which, when selected, present a list of years and months within a year. The calendar also contains a day button **830** for each day of the selected month and year. A user selects a particular date by choosing a year and month from drop down boxes **820** and **825**, and by pressing the day button for the desired day. The selected date is then input to the method described in **FIG. 9**.

[**0048**] **FIG. 9** is a flowchart for collecting data based on a Projected Bug Fix Date as done by component **220** of **FIG. 2**. The method starts with step **910** where the calendar based input screen of **FIG. 8** is presented to the user. Next, in step **915**, the system parses the date selected by the user. Step **920** reads from the software release database the schedule for software releases. Step **925** stores relevant data about each release in the data structure describing the software release. Step **930** builds a list of software releases.

[**0049**] Step **935** begins a loop that iterates over each software release in the list built in step **930**. Step **935** determines if unprocessed releases remain in the list. If at least one release remains then step **940** examines each data structure describing the software release to determine if the software release has a build date later than the date input in step **910**. If the software release does have a later date, then step **945** retrieves relevant data from the software release data structure and stores it in a release display data structure, preferably a hash table. The following step **947** obtains the next release, if any, in the list.

[**0050**] When all software releases in the list have been processed, step **935** directs the method to step **950**, which uses the release display data structure to dynamically generate HTML for a web page displaying a table of software release data similar to the table of **FIG. 10**.

[**0051**] **FIG. 10** shows a display of the data collected by the process of **FIG. 9**. Screen **1000** includes the query component selection index **310**, calendar **810**, and a release schedule table **1010** with each row representing an individual release and each column containing a data item describing the release. Column **1020** contains an identifying label that uniquely identifies a release. Column **1030** contains the date on which the next interim build will take place. Column **1040** contains the date on which the next throttle build will take place. Column **1050** contains the date by which software fixing the bug must be implemented in order to be included in the maintenance release build. Column **1060** contains the maintenance release date when the software release will be generally available to customers and others outside of the engineering organization.

[**0052**] **FIG. 11** is a flowchart for collecting data based on a Maintenance Release Identifier as done by component **225** of **FIG. 2**. First step **1115** preferably presents to the user a drop down box containing a list of major release identifiers, from which the user selects a major release identifier. Step **1125** parses the user's input into an input data structure. Step **1130** reads from the software release database the schedule for software releases. Next, step **1135** stores relevant data about each release in the data structure describing the software release. Step **1140** then examines each data structure describing the software release looking for a match based on the major release identifier. Each time a match is found, relevant data is taken from the data structure describing the software release and placed in a schedule display data structure, preferably a hash table. The method concludes in step **1150** where the schedule display data structure is used to dynamically generate HTML for displaying a web page containing a data collection from the release schedule database.

[**0053**] **FIG. 12** shows a display of the data collected by the process of **FIG. 11**. Schedule display screen **1200** includes the query component selection index **310** and a schedule table **1210** with each row representing a maintenance release within the major release identified by the release identifier obtained in step **1115** of **FIG. 11**. Column **1220** contains the label identifying the maintenance release. Column **1230** contains the date when the release was or will be first available for downloading to customers. Column **1240** contains the date when the release will be first included with hardware or distribution media produced during the manufacturing process for shipment to customers.

[**0054**] **FIG. 13** shows the initial screen **1300** of component Week **230** of **FIG. 2, a** display which allows a user to select a particular week of a year. Input to this component is a date corresponding to a week for which the user desires to obtain release information. Screen **1300** preferably comprises the query component selection index **310**, and a weekly calendar **1310** which contains a year drop down box **1320** and a month drop down box **1325** which, when selected, present a list of years and months within a year. The calendar also contains one day button **1330** for each Monday in the selected month. A user selects a particular week by choosing a year and month from drop down boxes **1320** and **1325**, and by pressing the day button for the Monday of the desired week. The selected date is then input to the method described in **FIG. 14**.

[0055] **FIG. 14** is a flowchart for collecting data based on an input of a particular week as done by Week **230** component of **FIG. 2**. First, step **1410** presents to the user a calendar based input screen of **FIG. 13** similar to the calendar described with reference to **FIG. 9** but only allowing selection of Mondays. Next, in step **1415**, the system parses the date selected by the user. Step **1420** reads from the software release database the schedule for software releases. Next, step **1425** stores relevant data about each software release in the data structure describing the software release. Step **1430** builds a list of software releases.

[0056] Step **1435** of **FIG. 14** begins a loop that iterates over each software release in the list built in step **1430** by determining if any unprocessed releases remain in the list. If at least one release remains in the list, then step **1440** examines each data structure describing the software release to determine if the software release has a build date during the week starting on the Monday input in step **1410**. If the software release does not have such a date, the method proceeds to step **1447**. If it does then step **1445** retrieves relevant data from the software release data structure and stores it in a release display data structure, preferably a hash table. Next, step **1447** obtains the next release, if any, in the list.

[0057] When all software releases in the list have been processed, step **1435** directs the method to step **1450** which uses the release display data structure dynamically to generate HTML for a web page that displays a table of software release data like **FIG. 15**.

[0058] **FIG. 15** shows a display of the data collected by the process of **FIG. 14**. Screen **1500** includes the query component selection index **310**, calendar **1310**, and a weekly release schedule table **1510** with each row representing an individual release and each column containing a data item describing the release. Columns **1520**, **1525** and **1530** may contain dates either in the past or the future depending on the date input by the user. Column **1515** contains an identifying label that uniquely identifies a release. Column **1520** contains a scheduled date for the interim build. If no interim build is scheduled for the specified week, column **1520** will be blank. If the user specifies a past date, the date in column **1520** will show the date that the interim build occurred. If the user specifies a future date, column **1520** will show the date the interim build is currently scheduled to occur. Column **1525** contains a date scheduled for a throttle build. Like column **1520**, if no throttle build is scheduled for the specified week, the column entry will be blank. If the date in column **1525** is in the past, it represents the date within the week specified by the user that the throttle build occurred. Otherwise, the date in column **1525** represents the date within the specified week that the throttle build is to occur. Column **1530** contains a date scheduled for a renumber build, which, if no renumber build is scheduled for the specified week, will be a blank entry. If the date in column **1530** is in the past, it represents the date within the specified week that the renumber build occurred. Otherwise, the date in column **1530** represents the date within the specified week that the renumber build is to occur.

[0059] **FIG. 16** shows the initial screen of **FIG. 2** component **235**, a release selection screen **1600** for allowing the selection of a particular release number. Screen **1600** comprises the release query component selection index **1610** as expanded by the method described with reference to **FIG. 5**, and interim build table **1620**. Release query component selection index **1610** contains a list of major release identifiers. After the user selects one of the major release identifiers, the system displays an interim build table **1620** organized left to right by columns **1630**. Each column represents a time series of interim build identifiers within the major release selected by the user. The bottom entry in each column represents a maintenance release identifier. Each entry in a column is preferably a selectable hyperlink that, when selected, presents information to the user about the bugs fixed in that release.

[0060] **FIG. 17** is a flowchart for collecting data based on the selection of a particular release identifier as done by **FIG. 2** Integrated Bug List **235**. The method starts with step **1710** when a user selects a major release identifier from the list of available major releases. Next, step **1715** uses HTML to dynamically generate an interim build table **1620 (FIG. 16)** and displays it to the user. In step **1720**, the user selects one of the entries representing a particular release in the interim build table, which causes step **1725** to parse the selection. Next, step **1730** uses the input from the selection in step **1725** as a key to query a release information database and find related information about the release. Finally, step **1735** uses HTML to dynamically generate a table of a selected subset of the information returned by the query, and displays a web page.

[0061] **FIG. 18** shows a display of the data collected by the process of **FIG. 17**. Integrated bug display **1800** comprises release query component selection index **1610** and interim build table **1620 (FIG. 16)**, and an integrated bug table **1810** with each row representing a particular release. Column **1820** contains an interim or maintenance release identifier which uniquely identifies a particular interim or maintenance release and preferably also contains in parenthesis a number representing the number of bugs fixed in the particular release. Column **1830** contains a list of bug identifiers that uniquely identify particular bugs that have been fixed in the release. Column **1840** contains a severity level corresponding to the bug with the identifier in column **1830**. Column **1850** contains a brief description of the bug identified by the bug identifier in column **1830**.

[0062] While the present invention has been described with reference to a preferred embodiment, those skilled in the art will recognize that various modifications may be made. Variations upon and modifications to the preferred embodiment are provided by the present invention, which is limited only by the following claims.

What is claimed is:

1. A method for a computer system to obtain software release information, comprising the steps of:

obtaining a set of defect data from a database having software defect information;

obtaining a set of schedule data from a database having software scheduling information;

obtaining a set of release data from a database having software release information;

relating data from at least two of said sets to create an organized data set; and

displaying the organized data set on an output device.

2. The method of claim 1 wherein the step of displaying includes dynamically generating a web page exhibiting the organized data set.

3. The method of claim 1 wherein said step of relating includes the steps of:

obtaining a first bug identifier from the set of defect data;

obtaining a second bug identifier and a first release identifier from the set of release data;

obtaining a second release identifier from the set of schedule data; and

identifying the defect data and the release data in which the first bug identifier equals the second bug identifier and the release data and the schedule data where the first release identifier equals the second release identifier.

4. The method of claim 1 wherein the step of displaying includes the steps of:

assigning a unique color to a first value;

deriving a second value from the organized data set; and

displaying an indicator having the unique color when the first value equals the second value.

5. The method of claim 1 wherein said output device is a remote display connected to a network.

6. The method of claim 1 wherein the step of obtaining a set of defect data filters the set of defect data by a defect identifier.

7. The method of claim 1 wherein the step of obtaining a set of schedule data filters the set of schedule data according to a date.

8. The method of claim 1 wherein the step of obtaining a set of release data filters the set of release data by a release identifier.

9. A system for obtaining software release information, comprising:

means for obtaining a set of defect data from a database having software defect information;

means for obtaining a set of schedule data from a database having software scheduling information;

means for obtaining a set of release data from a database having software release information;

means for relating data from at least two of said data sets to create an organized data set; and

means for displaying the organized data set on an output device.

10. The system of claim 9 wherein the means for displaying includes means for dynamically generating a web page exhibiting the organized data set.

11. The system of claim 9 wherein said means for relating includes:

means for obtaining a first bug identifier from the set of defect data;

means for obtaining a second bug identifier and a first release identifier from the set of release data;

means for obtaining a second release identifier from the set of schedule data; and

means for identifying the defect data and the release data in which the first bug identifier equals the second bug identifier and the release data and the schedule data where the first release identifier equals the second release identifier.

12. The system of claim 9 wherein the means for displaying includes:

means for assigning a unique color to a first value;

means for deriving a second value from the organized data set; and

means for displaying an indicator having the unique color when the first value equals the second value.

13. The system of claim 9 wherein said output device is a remote display connected to a network.

14. The system of claim 9 wherein the means for obtaining a set of defect data filters the set of defect data by a defect identifier.

15. The system of claim 9 wherein the means for obtaining a set of schedule data filters the set of schedule data according to a date.

16. The system of claim 9 wherein the means for obtaining a set of release data filters the set of release data by a release identifier.

17. A computer-useable medium embodying computer program code for causing a computer to obtain software release information by performing the steps of:

obtaining a set of defect data from a database having software defect information;

obtaining a set of schedule data from a database having software scheduling information;

obtaining a set of release data from a database having software release information;

relating data from at least two of said sets to create an organized data set; and

displaying the organized data set on an output device.

18. The computer-useable medium of claim 17 wherein said step of displaying includes dynamically generating a web page exhibiting the organized data set.

19. The computer-useable medium of claim 17 wherein said step of relating includes the steps of:

obtaining a first bug identifier from the set of defect data;

obtaining a second bug identifier and a first release identifier from the set of release data;

obtaining a second release identifier from the set of schedule data; and

identifying the defect data and the release data in which the first bug identifier equals the second bug identifier and the release data and the schedule data where the first release identifier equals the second release identifier.

20. The computer-useable medium of claim 17 wherein the step of displaying includes the steps of:

assigning a unique color to a first value;

deriving a second value from the organized data set; and

displaying an indicator having the unique color when the first value equals the second value.

21. The computer-useable medium of claim 17 wherein said output device is a remote display connected to a network.

22. The computer-useable medium of claim 17 wherein the step of obtaining a set of defect data filters the set of defect data by a defect identifier.

23. The computer-useable medium of claim 17 wherein the step of obtaining a set of schedule data filters the set of schedule data according to a date.

24. The computer-useable medium of claim 17 wherein the step of obtaining a set of release data filters the set of release data by a release identifier.

25. A system for obtaining software release information, comprising:

a memory storing computer instructions which cause a processing unit to

obtain a set of defect data from a database having software defect information,

obtain a set of schedule data from a database having software scheduling information,

obtain a set of release data from a database having software release information,

relate data from at least two of said sets to create an organized data set, and

display the organized data set on an output device; and

a processing unit coupled to the internal memory for executing said instructions.

26. The system of claim 25 wherein said instructions which cause a processing unit to display the organized data set further cause said processing unit to dynamically generate a web page exhibiting the organized data set.

27. The system of claim 25 wherein said instructions which cause a processing unit to relate data from at least two of said sets further cause said processing unit to

obtain a first bug identifier from the set of defect data,

obtain a second bug identifier and a first release identifier from the set of release data,

obtain a second release identifier from the set of schedule data, and

identify the defect data and the release data in which the first bug identifier equals the second bug identifier and the release data and the schedule data where the first release identifier equals the second release identifier.

28. The system of claim 25 wherein said instructions which cause a processing unit to display the organized data set further cause said processing unit to

assign a unique color to a first value,

derive a second value from the organized data set, and

display an indicator having the unique color assigned to the first value when the first value equals the second value.

29. The system of claim 25 wherein said output device is a remote display connected to a network.

30. The system of claim 25 wherein said instructions which cause a processing unit to obtain a set of defect data further cause said processing unit to filter the set of defect data by a defect identifier.

31. The system of claim 25 wherein said instructions which cause a processing unit to obtain a set of schedule data further cause said processing unit to filter the set of schedule data according to a date.

32. The system of claim 25 wherein said instructions which cause a processing unit to obtain a set of release data further cause said processing unit to filter the set of release data by a release identifier.

* * * * *